Local Decoding in Constant Depth

Shafi Goldwasser
MIT and Weizmann Institute
shafi@theory.csail.mit.edu

Dan Gutfreund
SEAS, Harvard University
danny@eecs.harvard.edu

Alexander Healy
SEAS, Harvard University
ahealy@fas.harvard.edu

Tali Kaufman
CSAIL, MIT
kaufmant@mit.edu

Guy N. Rothblum
CSAIL, MIT
rothblum@csail.mit.edu

Abstract

We study the complexity of locally decoding and locally list-decoding error correcting codes with good parameters. We focus on the circuit depth of the decoding algorithm and obtain new efficient constructions and tight lower bounds. our contributions are as follows:

1. We show general transformations from locally decodable and locally list-decodable codes with high-depth, say $\mathcal{NC}^1$, decoders, into codes with related parameters and constant depth $\mathcal{AC}^0$ decoders. These transformations use the delegation methodology introduced by Goldwasser *et al.* [STOC08]; the *decoding* algorithm is made more efficient by shifting some of its work to the *encoding* algorithm (in an error-robust way).
2. Using these transformations, we obtain explicit families of binary codes that are locally decodable or locally list decodable by *constant-depth* circuits of size polylogarithmic in the length of the codeword. For example, we build (i) an explicit binary code that is locally decodable in constant depth from constant (relative) distance, and (ii) a locally list-decodable binary code that decodes from relative distance $1/2-\varepsilon$ with list size at most $\text{poly}(1/\varepsilon)$. This second decoder is constant depth using majority gates of fan-in $\Theta(1/\varepsilon)$.
3. We show that our positive results are essentially tight. In particular, computing majority over $\Theta(1/\varepsilon)$ bits is essentially equivalent to locally list-decoding binary codes from relative distance $1/2-\varepsilon$ with list size at most $\text{poly}(1/\varepsilon)$. In fact, we show that a local-decoder for such a code can be used to construct a circuit of roughly the same size and depth that computes majority on $\Theta(1/\varepsilon)$ bits.
4. As a consequence of our positive result, and using the tight connection between locally list-decodable codes and average-case complexity, we obtain a new, more efficient, worst-case to average-case reduction for languages in $\mathcal{EXP}$.

Contents

1 Introduction

Error correcting codes are highly useful combinatorial objects that have found numerous applications both in practical settings as well as in many areas of theoretical computer science and mathematics. In the most common setting of error-correcting codes we have a message space that contains strings over some finite alphabet Γ (for simplicity we assume that all strings in the message space are of the same length). The goal is to design a function, which we call the *encoding function*, that encodes every message in the message space into a *codeword* such that even if a fairly large fraction of symbols in the codeword are corrupted it is still possible to recover from it the original message. The procedure that recovers the message from a possibly corrupted codeword is called *decoding*.

It is well known that beyond a certain fraction of errors, it is impossible to recover the original message, simply because the relatively few symbols that are not corrupted do not carry enough information to specify (uniquely) the original message. Still, one may hope to recover a list of candidate messages, one of which is the original message. Such a procedure is called *list-decoding*.

Typically, the goal of the decoder is to recover the entire message (or list of candidate messages) by reading the entire (possibly corrupted) codeword. There are settings, however, in which the codeword is too long to be read as a whole. Still, one may hope to recover any given individual symbol of the message, by reading only a small number of symbols from the corrupted codeword. This setting is called *local-decoding*, and both the unique and list decoding variants (as discussed above) can be considered. See Trevisan's survey [Tre04] for a more detailed discussion.

Locally decodable codes, both in the unique and list decoding settings, have found many applications in theoretical computer science, most notably in private information retrieval [CKGS98, KT00], and worst-case to average-case hardness reductions [STV01] (see below). Furthermore, they have the potential of being used for practical applications, such as reliably storing a large static data file, only small portions of which need to be read at a time.

Summary of contributions. Following the methodology laid out in [GGH+07] , we take a novel approach to the well-studied problem of efficient decoding: the encoder embeds information in the codeword that helps speed up the decoder's computation. In our new codes, the decoder uses the received word not only for reconstructing information about the original message, but also as a *computational resource*. We obtain:

1. A transformation from any locally decodable or locally-list decodable with a decoder in $\mathcal{NC}^1$, into a code with related parameters where the decoder is in $\mathcal{AC}^0$. This transformation is then used to obtain explicit codes with efficient decoders. These transformations are presented in Section 3.

2. An explicit binary code with polynomial rate and $\mathcal{AC}^0$ (probabilistic) local-decoding from a word that is corrupted in a constant fraction of locations (Theorem 1.3). This code has roughly the same parameters as the canonical example of a locally-decodable binary code with polynomial rate [STV01]. The construction is presented in Section 4.

3. An explicit family of locally list-decodable binary codes that are decodable from distance $1/2 - \varepsilon$ with list size $\mathrm{poly}(1/\varepsilon)$. The deocder is an $\mathcal{AC}^0$ circuit of size $\mathrm{poly}(\log M/\varepsilon)$ and with majority gates of size $\Theta(1/\varepsilon)$, where M is the length of the message (Theorem 1.5).

In particular, for $\varepsilon = 1/\text{polylog}\log(M)$ the decoder can be implemented in $\mathcal{AC}^0$ with size $\text{polylog}(M)$. This construction is presented in Section 4.

4. We show that the local list decoder above is optimal in its circuit depth. This is done by showing that *any* local list-decoder from distance $1/2-\varepsilon$ (with polynomial list size) can be used to copmute majority on $\Theta(1/\varepsilon)$ bits. In particular, using known lower bounds for computing majority, this means that constant-depth decoders cannot recover from large errors. The lower bound can be found in Section 5.

5. Finally, we discuss the consequences of our positive and negative results on local list-decoding to (fully black-box) hardness amplification for low complexity classes. See Section 6 for a full discussion.

Finally, we note that all previously known decoders for locally (list-)decodable codes with similar parameters compute majorities or finite field operations that (provably) cannot be implemented in $\mathcal{AC}^0$.

1.1 Background

We study the complexity of locally decoding and locally list decoding codes, we consider both the binary case (i.e. where the code's alphabet is $\{0,1\}$)[1] and the non-binary case.

Let us proceed more formally. Let $C : \Gamma^M \to \Gamma^N$ be the encoding function of an error-correcting code.[2] A local list-decoder D for a code C gets oracle access to a corrupted codeword, and outputs a "list" of ℓ local-decoding circuits $D_1, \ldots, D_\ell$. Each D_a is itself a probabilistic circuit with oracle access to the corrupted codeword. On input an index $j \in [M]$, a circuit D_a from the list tries to output the j-th symbol of the message. We say that the decoder is a (δ, ℓ)-local-list-decoder, if for every $y \in \{0,1\}^N$ and $m \in \{0,1\}^M$, such that the fractional Hamming distance between $C(m)$ and y is at most δ, with high probability at least one of the D_a's successfully decodes *every* symbol of the message y.[3] When the list size ℓ equals 1, i.e. there is a single decoding algorithm that is guaranteed to decode every bit of the message (w.h.p.), we say that the code is *locally decodable* (without a list). The quantity N/M which measures the amount of redundancy in the code is called *the rate* of the code.

Throughout this paper, when we speak of locally list decodable codes (rather than just locally decodable codes), we think of a local list-decoder as receiving an "advice" index $a \in [\ell]$, running D to output D_a, and then running D_a to retrieve the j-th message symbol. Note that by giving both D and the D_a's oracle access to the received word, and requiring them to decode individual symbols, we can hope for decoders whose size is much smaller than N (in particular we can hope for size that is poly-logarithmic in N). See section 2.2 for formal definitions of locally decodable and locally list decodable codes.

[1] Binary locally list decodable codes are of particular interest for their applications to hardness amplification.

[2] Formally, we consider a family of codes one for each message length M. The parameters listed above and below, e.g. N, ε, ℓ, should all be thought of as functions of M. For the exact definition of locally list-decodable codes see Definition 2.1.

[3] We would like to point out that for binary codes we use $(1/2 - \varepsilon, \ell)$ to denote the relative distance and list size, whereas previous work (e.g. [STV01]) used (ε, ℓ) to denote the same quantities (again, for binary codes). We find this notation more useful, especially when we work with non-binary codes.

1.2 This Work

This work studies the complexity of local and local-list decoders. Specifically, we ask whether it is possible to decode in constant depth. We answer this question in the affirmative, presenting transformations that reduce the depth of decoders and obtaining explicit locally decodable and locally list-decodable codes with constant-depth decoders.

Our approach. Building on the framework introduced in [GGH+08], we seek to improve the decoder's efficiency by delegating some of its computation to the encoder, in an error-robust way.

The idea of delegating computation from the decoder to the encoder seems dubious at first glance, as any information sent from the encoder might be corrupted by the noisy channel. This seems to leave the decoder very vulnerable to malicious behavior of the sender. To give the decoder a better guarantee, we ask more from the encoder: to send the results of the computations with redundancy that will allow the decoder to easily recover the correct results even from a corrupted word. This may seem to bring us back to square one; namely, the decoder again needs to decode a code. So where do we gain in efficiency? The key point is that we are not trying now to recover arbitrary information, but rather we are trying to recover the results of a certain *computation*. Here one can see a connection to program checking and correcting that we discuss below. Specifically, we show that if the decoder's computations have certain properties, which we discuss shortly, then the task of decoding the correct results of the computations can be done extremely efficiently – much more efficiently than the decoder's original computation. To develop our approach we look at functions that the decoder needs to compute and require them to have two properties:

1. (Random instance reduction) One can compute the function on any given instance by querying another function (say g) at a completely random location. This property allows us to correct the encoder's computations.

2. (Solved instance generator) We can generate efficiently a random instance of the function g together with g's value on this instance. This property allows us to check the correctness of (potentially corrupted) computations.

Combining these properties allows us to ensure (w.h.p.) that the decoder can recover from error introduced by the channel into the results of computations that were delegated to the encoder. Of course, this approach would not give us much if we could not show that the above properties can be implemented more efficiently than the original computations. To that end we show, using techniques that were developed in the field of cryptography [Bab87, Kil88, FKN94, IK02, AIK06], that for functions computable in $\mathcal{NC}^1$ these properties can be implemented in probabilistic constant parallel time. Thus, we can take any $\mathcal{NC}^1$ decoder and transform it into one that runs in constant parallel time or constant depth. We obtain transformations for binary locally decodacle codes and also for non-binary locally decodable and locally list decodable codes, the proofs are in Section 3:

Theorem 1.1 (Transformation for binary locally decodable codes)**.** *Let $C : \{0,1\}^M \to \{0,1\}^N$ be an explicit locally-decodable binary code that can be non-adaptively decoded from some constant distance $\delta < 1/4$ by a probabilistic $\mathcal{NC}^1$ circuit of size* polylog(N)*. Then there is an explicit binary code $C' : \{0,1\}^M \to \{0,1\}^{2N}$ that is locally-decodable from distance $\delta/2$ by a probabilistic $\mathcal{AC}^0$ circuit of size* polylog(N)*.*

Theorem 1.2. *(Tranformation for non-binary codes) Let Σ be a finite alphabet and let $C : \Sigma^M \to \Sigma^N$ be an explicit code that is non-adaptively locally list-decodable by probabilistic $\mathcal{NC}^1$ circuits from agreement ε and with list size ℓ. Then there is an explicit code $C' : \Sigma^M \to \Gamma^{N'}$ that is locally list-decodable by probabilistic $\mathcal{AC}^0$ circuits, from agreement ε and with list size 2ℓ, where $|\Gamma| = |\Sigma| \cdot O(1/\varepsilon)$, and $N' = \max\{N, 2^{(\log(N)/\varepsilon)^\delta}\}$, for arbitrarily small constant $\delta > 0$. In particular, for $\varepsilon \geq 1/\mathrm{poly}(\log N)$, we obtain $N' = N$. Finally, if C was uniquely locally decodable (with $\ell = 1$), then so is C'.*

Explicit Constructions. Using the above transformations, we obtain explicit codes that can be decoded very efficiently. Of course, to apply the transformations we need to begin with a decoder whose computations are in $\mathcal{NC}^1$! We design such codes, and in particular we obtain a binary locally decodable code, and both binary and non-binart locally list-decodable codes. These are described in the Theorems below, the proofs are in Section 4.

Theorem 1.3 (Locally decodable binary code)**.** *There is an explicit code $C : \{0,1\}^M \to \{0,1\}^{\mathrm{poly}(M)}$ that can be locally decoded from distance $1/25$ by probabilistic $\mathcal{AC}^0$ circuits of size $\mathrm{poly}(\log M)$.*

Theorem 1.4 (Locally list-decodable non-binary code)**.** *For every $\varepsilon > 0$, there is an explicit code $C : \{0,1\}^M \to \Sigma^N$ that is locally list-decodable by probabilistic $\mathcal{AC}^0$ circuits of size $\mathrm{poly}(\log M/\varepsilon)$ from agreement ε and with list size $\mathrm{poly}(1/\varepsilon)$. Where $|\Sigma| = 2^{\mathrm{poly}(1/\varepsilon)}$, and $N = M^{\mathrm{poly}(1/\varepsilon)}$.*

Theorem 1.5 (Locally list-decodable binary code)**.** *For every $2^{-\Theta(\sqrt{\log M})} \leq \varepsilon = \varepsilon(M) < 1/2$, there exists a $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \{0,1\}^{\mathrm{poly}(M)}\}_{M \in \mathbb{N}}$ with a local-decoder that can be implemented by a family of constant depth circuits of size $\mathrm{poly}(logM, 1/\varepsilon)$ using majority gates of fan-in $\Theta(1/\varepsilon)$ (and AND/OR gates of unbounded fan-in).*

Remark 1.6. *In the statement of Theorem 1.5, we view the decoder as a constant-depth circuit with majority gates of fan-in $\Theta(1/\varepsilon)$. This serves to demonstrate the connection between local list-decoding and computing majority. For distance bounded away from 1/2 by polylogarithmically small fraction, the code obtained is simply in $\mathcal{AC}^0$.*

Lower Bounds. Finally, we characterize the complexity of locally list-decoding binary codes. We show that the codes of Theorem 1.5 are essentially optimal. The proof of the following theorem is in section 5.

Theorem 1.7 (Informal)**.** *If there exists a binary code with a $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$-local-list-decoder of size s and depth d, then there exists a circuit of size $\mathrm{poly}(s)$ and depth $O(d)$ that computes majority on $\Theta(1/\varepsilon)$ bits.*

From Theorems 1.5 and 1.7 we conclude that computing the majority function on $\Theta(1/\varepsilon)$ bits is essentially equivalent to $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$-local-list-decoding binary codes: any circuit for a local-decoder of such a code can be used to construct a circuit of roughly the same size and depth that computes majority on $\Theta(1/\varepsilon)$ bits. In the other direction, there is an explicit $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$-locally-list-decodable code with a very efficient (in terms of size and depth) local-decoder that uses majority gates of fan-in $\Theta(1/\varepsilon)$.

By known lower bounds on the size of constant-depth circuits that compute majority [Raz87, Smo87], we obtain the following corollary.

Corollary 1.8 (Informal)**.** *Any constant-depth* $(1/2-\varepsilon, \mathrm{poly}(1/\varepsilon))$*-local list decoder for a binary code, must have size almost exponential in* $1/\varepsilon$*. This holds even if the decoder is allowed* $\mod q$ *gates, where* q *is an arbitrary prime number.*

In particular, we get an exact characterization of what is possible with constant-depth decoders: up to radius $1/2 - 1/\mathrm{poly}\log\log M$ locally-list-decodable codes with constant-depth decoders and good parameters exist, and beyond this radius they do not. We note that in fact we prove a stronger result in terms of the list size. We show that $(1/2-\varepsilon, \ell)$-local-list-decoding with a decoder of size s and depth d, implies a circuit of size $\mathrm{poly}(s, \ell)$ and depth d that computes majority on $O(1/\varepsilon)$ bits. This means that even if the list size is *sub-exponential* in $1/\varepsilon$, the size of the decoder still must be nearly exponential in $1/\varepsilon$ (even if the decoder is allowed $\mod q$ gates).

Hardness amplification. Hardness amplification is the task of obtaining from a Boolean function f that is somewhat hard on the average, a Boolean function f' that is very hard on the average. By a beautiful sequence of works [STV01, TV07, Tre03, Vio05], it is well known that there is a tight connection between binary locally (list) decodable codes and hardness amplification. Using this connection, we obtain both new positive and negative (in the spirit of Corollary 1.8) results on (black-box) hardness amplification procedures. We defer the statement of these results and a discussion to Section 6.

1.3 Related Work

Delegating Computation. Our approach for improving the decoder's efficiency by delegating some of its computation to the (possibly malicious) sender, is based on a delegation methodology and tools developed in a recent work of Goldwasser et al. on improving the efficiency of program checkers [GGH+08], and also inspired by the work of Applebaum, Ishai and Kushilevitz [AIK06] on improving the efficiency of cryptographic primitives. A discussion about the similarities and differences between these works and the results presented here follows.

The work in [GGH+08] develops a methodology of delegating computation as a way to increase the efficiency of program checkers (see [BK95]) and program testers/correctors (see [BLR93]). This idea plays a key role in their general approach of composing program checkers (and testers/correctors). We use similar ideas to improve the efficiency of decoders in error-correcting codes. These differences give rise to different challenges in the design of such protocols, and in particular in the implementation of the delegation methodology.

The fact that the prover is restricted to answering queries about the language being proved, in the case of program checkers testers and correctors, requires careful design of such protocols that typically use very specific properties of the functions being proved (checked). In our setting, the encoder can convey arbitrary information to the verifier. The challenge is for the encoder to relay to the decoder computationally helpful information over a noisy channel in an error-robust way. There is a guarantee, though, that the channel is not arbitrarily malicious and the noise rate is bounded, and the main challenge is then recovering from very large fractions of errors (especially in the list-decoding setting). This is also quite different from the setting of interactive proofs studied in [GGH+07].

The work of [AIK06] can also be viewed as improving the efficiency of players participating in a protocol by pushing computation from one of the participants to another (e.g. improving the efficiency of encryption at the expense of adding to the complexity of decryption). The main

difference between this approach and ours is that they consider protocols or objects in which the *goal* of the sender is to reveal the results of its computation to a receiver, so there is no issue of errors or of a malicious party. In our case, on the other hand, there is a noisy channel between the sender (encoder) and the receiver (decoder) whose efficiency we want to improve. Given these differences, the tools we use are somewhat different from those used in [AIK06]. Nonetheless, some of the techniques we employ are similar.

Local Decoding. It is well known that for every (non-trivial) $(1/2-\varepsilon,\ell)$-locally-list-decodable code, it must hold that $\ell = \Omega(1/\varepsilon^2)$ [Bli86, GV05] (in fact this bound holds even for standard, non-local, list decoding). Thus, aiming to stay within polynomial factors of the best possible information theoretic parameters, our primary goal is to understand the complexity of decoding $(1/2-\varepsilon, \mathrm{poly}(1/\varepsilon))$-locally-list-decodable binary codes that have polynomial rate (i.e. where $N(M) = \mathrm{poly}(M)$). We consider such codes to have "good" parameters (we elaborate on this choice below).

An explicit code with good parameters was given by Sudan, Trevisan and Vadhan [STV01]. The local-decoder for this code (namely the algorithm D as well as the circuits D_a) is in the complexity class NC^2 (i.e. its depth is poly-logarithmic in its input length). Explicit codes with local-decoders in the (strictly lower) class AC^0 (i.e. constant depth unbounded fan-in decoders) are also known [GL89]. However, these codes do not have good parameters.[4] Specifically the Hadamard code has such a decoder [GL89], but its rate is exponential in M.

Lower Bounds for Local List-Decoding. The question of lower bounding the complexity of local-list-decoders was raised by Viola [Vio06]. He conjectured that $(1/2-\varepsilon,\ell)$-locally-list-decodable codes require computing majority over $O(1/\varepsilon)$ bits,[5] even when the list size ℓ is exponential in $1/\varepsilon$. Note that while exponential lists are not commonly considered in the coding setting (the focus instead is on polynomial or even optimal list sizes), they do remain interesting for applications to (non-uniform) worst-case to average-case hardness reductions. In particular, lower bounds for local-list-decoding with exponential lists, imply impossibility results for *non-uniform* black-box worst-case to average-case hardness reductions (see Section 6). In this paper we prove the conjecture for the case of sub-exponential size lists. While a proof of the full-blown conjecture remains elusive, there are results for other (incomparable) special cases:

Viola [Vio06] gave a proof (which he attributed to Madhu Sudan) of the conjecture for the special case of the standard *non-local* list-decoding setting. It is shown that a list-decoder from distance $1/2-\varepsilon$ can be used to compute majority on $\Theta(1/\varepsilon)$ bits, with only a small blow-up in the size and depth of the decoder. This result rules out, for example, constant-depth list-decoders whose size is $\mathrm{poly}(1/\varepsilon)$. Note, however, that in the non-local list decoding setting the size of the decoder is at least N (the codeword length) because it takes as input the entire (corrupted) codeword. This means that the bound on the size of constant-depth decoders does not have consequences for fairly large values of ε. For example, when $\varepsilon \geq 1/\log N$, the only implication that we get from [Vio06], is that there is a constant-depth circuit of size at least $N \geq 2^{1/\varepsilon}$ that computes majority on instances of size $1/\varepsilon$. But this is trivially true, and thus we do not get any contradiction. In the local-decoding

[4] We note that for non-binary codes, i.e. codes with large alphabets, one can construct codes with constant-depth local list-decoders and "good" parameters, see [GGH+07].

[5] By "require" we mean that the decoding circuit can be used to construct a circuit of comparable size and depth that computes the majority function on $O(1/\varepsilon)$ bits.

setting the decoders' circuits are much smaller and thus we can obtain limitations for much larger ε's. Indeed in this paper we rule out constant-depth decoders for $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$-local-list-decoders for any ε smaller than $1/\mathrm{poly}\log\log N$ (and recall that this matches the construction of [GGH+07]).

Viola [Vio06] also proved that there are no constant-depth decoders (with polynomial-size lists) for *specific* codes, such as the Hadamard and Reed-Muller codes. We, on the other hand, show that there are no such decoders for *any* code (regardless of the code's rate, and even with sub-exponential list size).

Recently (independently of our work), Shaltiel and Viola [SV08] gave a beautiful proof of the conjecture for the local-decoding setting, with ℓ exponential in $1/\varepsilon$, but for the special case that the decoder is restricted to have *non-adaptive* access to the received word. (I.e., they give a lower bound for decoders that make all their queries to the received word simultanuously.) Our result is incomparable to [SV08]: we prove Viola's conjecture only for the case that ℓ is sub-exponential in $1/\varepsilon$, but do so for *any* decoder, even an adaptive one. We emphasize that for important ranges of parameters the best codes known to be decodable in constant depth use *adaptive* decoders. In particular, the constant depth decoder of [GGH+07], as well as its improvement in this work, are adaptive. In light of this, it is even more important to show lower bounds for adaptive decoders.

1.4 On the Choice of Parameters

In this work binary codes with polynomial-rate are considered to have "good" parameters. Usually in the standard coding-theory literature, "good" codes are required to have *constant* rate.[6] We note that, as far as we know, there are no known locally-decodable codes (both in the unique and list decoding settings) with constant rate (let alone codes that have both constant rate *and* have decoders that are in the low-level complexity classes that we consider here). The best binary locally decodable codes known have polynomial rate [STV01]. It is an interesting open question to find explicit codes with constant or even polylogarithmic rate.

Finally, we note that in this work we do not (explicitly) consider the query complexity of the decoder. The only bound on the number of queries the decoder makes to the received word comes from the bound on the size of the decoding circuit. The reason is that known codes with much smaller query complexity than the decoder size (in particular constant query complexity) have a very poor rate (see e.g. [Yek08]). Furthermore, there are negative results that suggest that local-decoding with small query complexity may *require* large rate [KT00, DJK+02, Oba02, KdW04, WdW05, GKST06].

[6] We do remark that for applications such as worst-case to average-case reductions, polynomial or even quasi-polynomial rates suffice.

2 Preliminaries

For a string $x \in \Sigma^*$ (where Σ is some finite alphabet) we denote by $|x|$ the length of the string, and by x_i or $x[i]$ the i'th symbol in the string. For a finite set S we denote by $y \in_R S$ that y is a uniformly distributed sample from S. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \ldots, n\}$. For a finite alphabet Γ we denote by Δ_Γ the relative (or fractional) Hamming distance between strings over Γ. That is, let $x, y \in \Gamma^n$ then $\Delta_\Gamma(x, y) = \Pr_{i \in_R [n]}[x[i] \neq y[i]]$, where $x[i], y[i] \in \Gamma$. Typically, Γ will be clear from the context, we will then drop it from the subscript.

2.1 Circuit and Complexity Classes

We assume that the reader is familiar with standard complexity classes such as $\mathcal{NP}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$. For a positive integer $i \geq 0$, $\mathcal{AC}^{\mathsf{i}}$ circuits are Boolean circuits (with AND, OR and NOT gates) of size $\mathrm{poly}(n)$, depth $O(\log^i n)$, and unbounded fan-in AND and OR gates (where n is the length of the input). $\mathcal{NC}^{\mathsf{i}}$ circuits are boolean circuits of size $\mathrm{poly}(n)$ and depth $O(\log^i n)$ where the fan-in of AND and OR gates is 2. We use the same notations to denote the classes of functions computable by these circuit models. We denote by $\mathcal{AC}$ the class $\bigcup_{i \in \mathbb{N}} AC^i$, and by $\mathcal{NC}$ the class $\bigcup_{i \in \mathbb{N}} NC^i$. $\mathcal{RNC}^{\mathsf{i}}$, $\mathcal{RAC}^{\mathsf{i}}$, $\mathcal{RNC}$ and $\mathcal{RAC}$ are the (one-sided) randomized analogs of the above classes. In particular, $\mathcal{AC}^{\mathsf{0}}$ circuits are boolean circuits (with AND, OR and NOT gates) of constant-depth, polynomial size, and unbounded fan-in AND and OR gates. $\mathcal{NC}^{\mathsf{1}}$ circuits are boolean circuits of fan-in 2, polynomial size and logarithmic (in the input size) depth. $\mathcal{NC}^{\mathsf{0}}$ circuits are similar to $\mathcal{NC}^{\mathsf{1}}$, but have constant-depth. Note that in $\mathcal{NC}^{\mathsf{0}}$ circuits, every output bit depends on a constant number of input bits. $\mathcal{AC}^{\mathsf{0}}$, $\mathcal{NC}^{\mathsf{1}}$ and $\mathcal{NC}^{\mathsf{0}}$ are the classes of languages (or functions) computable (respectively) by $\mathcal{AC}^{\mathsf{0}}/\mathcal{NC}^{\mathsf{1}}/\mathcal{NC}^{\mathsf{0}}$ circuits. $AC^i[q]$ (for a prime q) are similar to $\mathcal{AC}^{\mathsf{i}}$ circuits, but augmented with $\mod q$ gates.

Throughout, circuits may have many output bits (we specify the exact number when it is not clear from the context). Also, often we consider uniform circuit classes. Unless we explicitly note otherwise, circuit families are log-space uniform, i.e. each circuit in the family can be described by a Turing machine that uses a logarithmic amount of space in the size of the circuit (for non-polynomial size circuit families the default uniformity is using space logarithmic in the circuit family size). Thus $\mathcal{NC}^{\mathsf{0}}$ (resp. $\mathcal{NC}^{\mathsf{1}}$) computations in this work are equivalent to constant (resp. logarithmic) parallel time in the CREW PRAM model.

Finally, we extensively use oracle circuits: circuits that have (unit cost) access to an oracle computing some function. We sometimes interpret this function as a string, in which case the circuit queries and index and receives from the oracle the symbol in that location in the string.

2.2 Locally Decodable and Locally List-Decodable Codes

The definition of locally list-decodable codes follows the formulation of Sudan, Trevisan and Vadhan [STV01] (with some small modifications). (Uniquely) locally decodable codes are defined as a special case (with list size 1).

Definition 2.1 (Locally decodable and locally list-decodable codes)**.** Let Γ be a finite alphabet. An ensemble of functions $\{C_M : \{0,1\}^M \to \Gamma^{N(M)}\}_{M \in \mathbb{N}}$ is a $(\delta(M), \ell(M))$-*locally-list-decodable code*, if there is an oracle Turing machine $D[\cdot, \cdot, \cdot, \cdot]$ that takes as input an index $i \in [M]$, an "advice"

string $a \in [\ell(M)]$ and two random strings r_1, r_2,[7] and the following holds: for every $y \in \Gamma^{N(M)}$ and $x \in \{0,1\}^M$ such that $\Delta_\Gamma(C_M(x), y) \leq \delta(M)$,

$$\Pr_{r_1}\left[\exists a \in [\ell] \text{ s.t. } \forall i \in [M] \ \Pr_{r_2}[D^y(a, i, r_1, r_2) = x[i]] > 3/4\right] > 3/4 \tag{1}$$

If $|\Gamma| = 2$ we say that the code is binary. If $\ell = 1$ we say that the code is a *(uniquely) locally decodable code*. We say that the code is explicit if C_M can be computed in time $\text{poly}(N(M))$.

Remark 2.2. *One should think of the decoder's procedure as having two stages: first it tosses coins r_1 and generates a sequence of ℓ circuits $\{F_a(\cdot,\cdot)\}_{a\in[\ell]}$, where $F_a(i, r_2) = D(a, i, r_1, r_2)$. In the second stage, it uses the advice a to pick the probabilistic circuit F_a and use it (with randomness r_1) to decode the message symbol at index i. In [STV01] the two-stage process is part of the definition, for us it is useful to encapsulate it in one machine (D).*

In the sequel it will be convenient to simplify things by ignoring the first stage, and consider D as a probabilistic circuit (taking randomness r_2) with two inputs: the advice a and the index to decode i, with the property that (always) for at least one $a \in [\ell]$, $D(a,\cdot)$ decodes correctly every bit of the message (with high probability over r_2). Indeed if we hardwire any "good" r_1 (chosen in the first stage) into D then we are in this situation. This happens with probability at least 99/100. Thus in our proofs we will assume that this is the case, while (implicitly) adding 1/100 to the bound on the overall probability that the decoder errs. This simplification makes our proofs much clearer (since we do not have to deal with the extra randomness r_1).

Remark 2.3 (Complexity of decoding)**.** *We often restrict the complexity of D. When we say, for example, that C is locally list-decodable (with the specified parameters) by $\mathcal{AC}^0$ circuits, we mean that D (and all of the circuits $\{F_a\}$) are probabilistic $\mathcal{AC}^0$ circuits of size* $\text{poly}(\log(N)/\varepsilon)$.

As a stepping stone in our constructions we will also use a relaxed variant: approximate locally-decodable codes, see Trevisan [Tre03].

Definition 2.4. [approximate locally list-decodable codes [Tre03]] Let Γ be a finite alphabet. We say that a code $\{C_M : \{0,1\}^M \to \Gamma^{N(M)}\}_{M\in\mathbb{N}}$ is δ-approximate (d,ℓ)-locally-list-decodable, if it is the same as in Definition 2.1 with the following relaxation of (2.1):

$$\Pr_{r_1}\left[\exists a \in [\ell] \text{ s.t. } \Pr_{i\in_R[M]}\left[\Pr_{r_2}[D^y(a, i, r_1, r_2) = x[i]] > 9/10\right] \geq 1-\delta\right] > 99/100$$

Less formally, in approximate codes the requirement is that for at least one advice string, the decoder decodes at least a $1-\delta$ fraction of the bits of the message (but not necessarily all the bits of the message as in Definition 2.1). In our description of the decoders, we will use the two-stage process view of this definition as discussed in Remark 2.2.

2.3 Majority and Related Functions

We frequently use the two following results of Ajtai [Ajt93] (which are based on [AB84]).

[7] The length of these random strings lower-bounds D's running time. Later in this work, when we consider D's with bounded running time, the length of these random strings will also be bounded.

Lemma 2.5. *For all constants $c > 0$, there is a family of $\mathcal{AC}^0$ circuits that approximates the weight of x to within a factor of $1 \pm 1/\log^c(n)$: i.e., given $x \in \{0,1\}^n$, output a value δ such that if the fraction of 1's in x is δ', then $(1 - 1/\log^c(n))\delta' \leq \delta \leq (1 + 1/\log^c(n))\delta'$.*

In particular,

Lemma 2.6. *For all constants $c > 0$, there is a family of $\mathcal{AC}^0$ circuits that computes the following approximate majority promise problem: given $x \in \{0,1\}^n$, decide whether the fraction of 1's in x is greater than $1/2 + 1/\log^c(n)$ or less than $1/2 - 1/\log^c(n)$ (and give arbitrary answer if none of the two is the case).*

We use the promise problem Π, defined in [Vio06] as follows:
$\Pi_{Yes} = \{x : x \in \{0,1\}^{2k}$ for some $k \in \mathbb{N}$ and $weight(x) \leq k - 1\}$
$\Pi_{No} = \{x : x \in \{0,1\}^{2k}$ for some $k \in \mathbb{N}$ and $weight(x) = k\}$
where $weight(x)$ is the number of bits in x which are 1.

We will extensively use the fact, proven in [Vio06], that computing the *promise* problem Π on $2k$ bit inputs is (informally) "as hard" (in terms of circuit depth) as computing majority of $2k$ bits. This is stated formally in the claim below:

Claim 2.7 ([Vio06])**.** *Let $\{C\}_{M \in \mathbb{N}}$ be a circuit family of size $S(M)$ and depth $d(M)$ that solves the promise problem Π on inputs of size M. Then, for every $M \in \mathbb{N}$, there exists a circuit B_M of size $\mathrm{poly}(S(M))$ and depth $O(d(M))$ that computes majority on M bits. The types of gates used by the B_M circuit are identical to those used by C_M. E.g., if C_M is an $AC^0[q]$ circuit, then so is B_M.*

2.4 Randomized images

We start by defining the properties of functions and languages that we need for our approach. The first property says that we can easily generate a random instance together with the evaluation of the function on this input.

Definition 2.8 (Solved instance generator)**.** Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function. We say that a randomized algorithm G is a *solved instance generator* for f if, given 1^n, it generates a pair (x, y), where x is a uniformly random element of $\{0,1\}^n$ and $y = f(x)$.

The second property is a reduction from one function to another that says, roughly, that we can evaluate the first function on every instance by querying the second function in a random location.

Definition 2.9 (Random instance reduction)**.** Let $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \to \{0,1\}^*$ be two functions. We say that a pair of algorithms $(\mathcal{R}, \mathcal{E})$ is a *random instance reduction* from f to g if $\mathcal{R}$ is a randomized algorithm that given $x \in \{0,1\}^n$, generates a pair (x', τ), where x' is a uniformly random element of $\{0,1\}^{m(n)}$ and $\tau \in \{0,1\}^*$ and it holds that $\mathcal{E}(g(x'), \tau) = f(x)$.

If $m(n) = n$ we say that the random instance reduction is *length-preserving*. If f and g are the same function, we say that it is a random instance self-reduction.[8] We call $\mathcal{R}$ the *Randomizer* and $\mathcal{E}$ the *Evaluator*.

[8] Random instance self-reductions are a special form of what is called in the literature *random self-reductions*. The word instance in our terminology, should emphasize the fact that the reduction is from one instance to another (random) instance. General random self-reductions can make many *self*-queries to the function in order to compute its value on a given instance.

The objects that we will be interested in are pairs of functions that have the above two properties.

Definition 2.10 (Randomized image)**.** Let $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \to \{0,1\}^*$ be two functions. We say that g is a *randomized image* of f, if there is a random instance reduction from f to g, and g has a solved instance generator.

We say that it is length-preserving if the random instance reduction is length-preserving, and that it is a randomized self-image if $f = g$. Finally, we say that the randomized image can be implemented in some complexity class $\mathcal{C}$, if the algorithms $G, \mathcal{R}$ and $\mathcal{E}$ (from Definitions 2.8 and 2.9) can be implemented in this class.

Next we present several languages that have extremely efficient randomized images. In the sequel we will abuse the term by saying that a language has a randomized image, meaning that its characteristic function has a randomized image. Our constructions use techniques that were developed in the field of cryptography, with some appropriate modifications.

One important family of functions that have randomized images are word problems over finite groups.

Definition 2.11. Let $(G, \odot)$ be a group. We define the word problem of G to be the function $L_G : G^* \to G$, where $L_G(a_1, \ldots, a_n) = a_1 \odot a_2 \odot \cdots \odot a_n$.

Claim 2.12. *Let $(G, \odot)$ be a finite group. Then L_G has a length-preserving randomized self-image that can be implemented by $\mathcal{NC}^0$ circuits given that they can sample random elements from G.*

Proof. Our constructions are based on a randomization technique from [Bab87, Kil88]. Details follow.

Solved instance generator: We would like to sample a random instance $x \in G^n$ together with $y = L_G(x)$. Given 1^n and a random string $\bar{a} = (a_1, \ldots, a_n) \in_R G^n$, define x to be $(a_1, a_1^{-1} \odot a_2, a_2^{-1} \odot a_3, \ldots, a_{n-1}^{-1} \odot a_n)$, and y to be a_n. Since $a_1, \ldots, a_n$ are independently and uniformly distributed, so are $a_1, a_1^{-1} \odot a_2, a_2^{-1} \odot a_3, \ldots, a_{n-1}^{-1} \odot a_n$. Clearly, $L_G(x) = a_1 \odot a_1^{-1} \odot a_2 \odot a_2^{-1} \odot a_3 \cdots a_{n-1}^{-1} \odot a_n = a_n$. Finally, note that every element in x' is a function of two elements in $\bar{a}$, therefore the procedure can be implemented by an $\mathcal{NC}^0$ circuit over the alphabet G.

Random instance self-reduction: The randomizer $\mathcal{R}$, given $x \in G^n$ and a random string $\bar{a} = (a_1, \ldots, a_n) \in_R G^n$, outputs $x' \in G^n$ which is $(x_1 \odot a_1, a_1^{-1} \odot x_2 \odot a_2, a_2^{-1} \odot x_3 \odot a_3, \ldots, a_{n-1}^{-1} \odot x_n \odot a_n)$, and τ which is a_n^{-1}. Define $\mathcal{E}(\sigma, \tau) = \sigma \odot \tau$. Since $a_1, \ldots, a_n$ are independently and uniformly distributed, so are $x_1 \odot a_1, a_1^{-1} \odot x_2 \odot a_2, a_2^{-1} \odot x_3 \odot a_3, \ldots, a_{n-1}^{-1} \odot x_n \odot a_n$. Clearly, $\mathcal{E}(f(x'), a_n^{-1}) = x_1 \odot a_1 \odot a_1^{-1} \odot x_2 \odot a_2 \odot a_2^{-1} \odot x_3 \odot a_3 \cdots a_{n-1}^{-1} \odot x_n \odot a_n \odot a_n^{-1} = x_1 \odot x_2 \odot \cdots \odot x_n = L_G(x)$. Finally, note that every element in y is a function of one element in x and two elements in $\bar{a}$, therefore $\mathcal{R}$ can be implemented by an $\mathcal{NC}^0$ circuit over the alphabet G ($\mathcal{E}$ is over a finite domain so it is clearly in $\mathcal{NC}^0$). ∎

Corollary 2.13. *The parity function: $Parity(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$ where $x_i \in \{0,1\}$ and the sum is over $GF(2)$ has a length-preserving randomized self-image that can be implemented by $\mathcal{NC}^0$ circuits.*

Barrington [Bar89] has shown that L_{S_5} is complete for the class $\mathcal{NC}^1$ under $\mathcal{NC}^0$ reductions[9] (S_5 is the symmetric group over five elements). We conclude:

[9] While [Bar89] only considers $\mathcal{AC}^0$ reductions, it is clear that the reduction is in fact $\mathcal{NC}^0$ as every element of S_5 in the resulting word problem depends on exactly one input bit of the original instance.

Corollary 2.14. *There is an $\mathcal{NC}^1$-complete function under $\mathcal{NC}^0$ reductions that has a length-preserving randomized self-image that can be implemented by $\mathcal{NC}^0$ circuits* that are given access to a source of random elements in S_5.

Having access to a source of random elements in S_5 is a non-standard requirement. The reason that we need it is that standard (Boolean) $\mathcal{NC}^0$ circuits cannot sample uniformly from a set of size $|S_5| = 120$ (given access to a source of random bits). This motivates (as in [AIK06]) the following construction, based on ideas from [IK02], which gives complete languages in the (higher) class $\oplus$-$\mathcal{L}$ that have efficient randomized images that can be implemented by *Boolean* $\mathcal{NC}^0$ circuits. Consider the following language:

Definition 2.15. The language CMD (for Connectivity Matrix Determinant[10]) is defined as follows: an instance of the language is a $n \times n$ matrix A that has 0/1 entries on the main diagonal and above it, -1 on the second diagonal (one below the main), and 0 below this diagonal. A is represented by the list of $\frac{n(n+1)}{2}$ 0/1 entries on and above the main diagonal. Define the characteristic function of L to be $det(A)$ where the determinant is computed over $GF(2)$.

Claim 2.16. *[IK02] CMD is $\oplus$-$\mathcal{L}$-complete under $\mathcal{NC}^0$ reductions.*

Next, building on ideas from [Bab87, Kil88, FKN94, IK02, AIK06], we show that CMD has a a randomized self-image that can be implemented in $\mathcal{AC}^0[\oplus]$. We will then modify the language to obtain a randomized image (but not a self-image) for CMD that can be implemented in $\mathcal{NC}^0$.

Claim 2.17. *CMD has a length-preserving randomized self-image that can be implemented by $\mathcal{AC}^0[\oplus]$ circuits.*

Proof. For every integer n, Consider the family of $n \times n$ matrices $\mathcal{M}_1$ with 1's on the main diagonal, 0's below it and 0/1 above it. Similarly consider the family of $n \times n$ matrices $\mathcal{M}_2$ with 1's on the main diagonal, 0/1 in the last column (except for entry (n, n) which is 1), and 0's in all other entries. Notice that every matrix in $\mathcal{M}_1 \cup \mathcal{M}_2$ has determinant 1. It is shown in [IK02] that for every instance $A \in CMD$ (respectively $A \notin CMD$), the matrix $C_1 \times A \times C_2$ (represented by the entries on and above the main diagonal) is uniformly distributed over the elements in CMD (respectively not in CMD) of the same length, when C_1 and C_2 are uniformly distributed in $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively. Furthermore, CMD halves the space, that is, for every input length, exactly half the instances are in CMD.

Given these properties, we can construct a solved instance generator and a random instance self-reduction for (the characteristic function of) CMD. The solved instance generator on 1^n chooses at random $b \in_R \{0, 1\}$ and constructs the matrix H_b that has -1 on the second diagonal, b in entry $(1, n)$ and 0's elsewhere. It then chooses $C_1 \in_R \mathcal{M}_1$ and $C_2 \in_R \mathcal{M}_2$ and outputs the pair $(C_1 \times H_b \times C_2, b)$. By the above properties $L(x) = det(C_1 \times H_b \times C_2) = b$, and $C_1 \times H_b \times C_2$ is uniformly distributed (over the choice of b, C_1, C_2).

Notice that,

$$(C_1 \times H_b \times C_2)_{ij} = \sum_{\ell=1}^{n} \sum_{k=1}^{n} ((C_1)_{ik} \cdot (H_b)_{k\ell} \cdot (C_2)_{\ell j}) \tag{2}$$

[10]The origin of the name comes from the fact that the determinant of matrices we consider represents information about the number of of paths between two nodes in a directed acyclic graph.

Thus every entry in the output of the solved instance generator is a degree 3 polynomial over $GF(2)$ of the bits representing C_1, C_2 and H_b and it is therefore computable by $\mathcal{AC}^0[\oplus]$ circuits.

For the random instance self-reduction, given an $n \times n$ matrix A as above, the randomizer $\mathcal{R}$ chooses $b \in_R \{0,1\}$ and constructs the matrix A' which is equal to A everywhere except for entry $(1,n)$ defined to be $A'_{1,n} = A_{1,n} \oplus b$. Next $\mathcal{R}$ chooses $C_1 \in_R \mathcal{M}_1$ and $C_2 \in_R \mathcal{M}_2$ and outputs the pair $(C_1 \times A' \times C_2, b)$. Finally, for $\mathcal{E}$ to retrieve the value of $det(A)$ from $det(C_1 \times A' \times C_2)$ and b, we notice that $det(A) = det(A') \oplus b$ (when the determinants are computed over $GF(2)$). This is because the entry $(1,n)$ appears in exactly one summand in the determinant of A and A'. So $det(C_1 \times A' \times C_2) \oplus b = det(A') \oplus b = det(A)$. As in the case of the solved instance generator, the random instance self-reduction is computable in $\mathcal{AC}^0[\oplus]$. ■

As in [AIK06], the reason the above construction is not in $\mathcal{NC}^0$, is that every entry in the product matrix is a sum (over $GF(2)$) of many elements, each is a product (over $GF(2)$) of three elements. Thus it does not depend on a constant number of input and randomness bits. To overcome this, as in [AIK06], we randomize this sum using the randomizing technique for the word problem over the group Z_2^+ (Corollary 2.12).

Let us start by defining the following variant of CMD.

Definition 2.18. the language $DCMD$ (for Decomposed Connectivity Matrix Determinant) is defined as follows: for every integer $n > 0$, an instance (of length $\frac{n^3(n+1)}{2}$) of $DCMD$ is described by $\frac{n(n+1)}{2}$ n^2-tuples of bits. The n^2-tuples are indexed by $1 \le j \le i \le n$. The (i,j)'th tuple is denoted $(b_{i,j}^{(1)}, \ldots, b_{i,j}^{(n^2)})$. We think of such an instance as a $n \times n$ matrix A with -1 on the second diagonal, 0's below the second diagonal, and the $\frac{n(n+1)}{2}$ entries above the second diagonal are given by

$$A_{i,j} = \bigoplus_{k=1}^{n^2} b_{i,j}^{(k)}$$

We say that such an instance is in $DCMD$ if and only if the determinant over $GF(2)$ of the matrix it represents is 1.

Claim 2.19. *$DCMD$ is a randomized image of CMD that can be implemented by $\mathcal{NC}^0$ circuits.*

Proof. We describe the solved instance generator, the random instance reduction is obtained similarly. The first step of the solved instance generator is as in the proof of Claim 2.17. That is, let $A = C_1 \times H_b \times C_2$ where $b \in_R \{0,1\}$, $C_1 \in_R \mathcal{M}_1$ and $C_2 \in_R \mathcal{M}_2$. Now consider the entries of A as given in Equation 2. For every entry, (i,j), the solved instance generator chooses n^2 random bits, $(b_{i,j}^{(1,1)}, \ldots, b_{i,j}^{(n,n)})$. It then generates the $DCMD$ instance whose (k,ℓ)'th entry (here we change to indexing in $[n] \times [n]$ instead of $[n^2]$) in the (i,j)'th n^2-tuple is

$$b_{k,\ell-1}^{i,j} \oplus (C_1)_{ik} \cdot (H_b)_{k\ell} \cdot (C_2)_{\ell j} \oplus b_{i,j}^{(k,\ell)}$$

Where we define $b_{i,j}^{(k,0)} = b_{i,j}^{(k-1,n)}$, and $b_{i,j}^{(0,n)} = b_{i,j}^{(n,n)} = 0$ (note that we do not use our random choice for $b_{i,j}^{(n,n)}$, it is included here only to simplify the indexing).

by applying the argument of Claim 2.12 w.r.t. the group Z_2^+ on each entry, we see that this instance is a $DCMD$ representation of the matrix A. Furthermore, For every entry on and above

the main diagonal of A, the n^2-tuple associated with it is uniformly distributed among all tuples whose parity equals to that entry. Since the value of every entry on and above the main diagonal is uniformly and independently distributed (as argued in the proof of Claim 2.17), we obtain a uniformly distributed instance of $DCMD$ whose determinant is b. Finally, we note that every bit of the $DCMD$ instance is a function of five input/randmoness bits, and thus the solved instance generator can be implemented by an $\mathcal{NC}^0$ circuit. ■

Now that we have randomized images for several complete problems, we observe that the property is preserved by reductions.

Claim 2.20. *If a language L is hard for some complexity class $\mathcal{C}_1$ under some class $\mathcal{C}_2$ of Karp-reductions and if L has a randomized image that can be implemented in $\mathcal{C}_2$, then every language $L' \in \mathcal{C}_1$ has a randomized image that can be implemented in $\mathcal{C}_2$.*

Proof. The randomized image of L' will be the one of L. To obtain a random instance reduction, given an instance x for the language L', reduce it to an instance y for L. Then apply on y the random instance reduction from L to its randomized image. ■

Another useful property of functions is the following strong form of downward self-reducibility.

Definition 2.21. We say that a language L is *strongly downward self-reducible* if, for every constant $\delta > 0$, L can be decided by a family of polynomial-size constant-depth oracle circuits, such that the circuit for length n makes queries to an oracle that solves L at input length n^δ.

Using associativity of group operations, the following is straightforward.

Claim 2.22. *Let $(G, \odot)$ be a finite group. Then L_G (the word problem over G) is strongly downwards self reducible.*

We conclude this section with the following lemma which says that there is an $\mathcal{NC}^1$-complete language that has all the properties that we need.

Lemma 2.23. *There is an $\mathcal{NC}^1$-complete language under $\mathcal{NC}^0$ reductions, that is strongly downwards self-reducible, and has a randomized image that can be implemented by $\mathcal{NC}^0$ circuits.*

Proof. Barrington [Bar89] showed that the decisional version of L_{S_5} (in which one has to decide whether the resulting permutation in S_5 is the identity or not) is $\mathcal{NC}^1$ complete under $\mathcal{NC}^0$ reductions. By Claim 2.22 it is strongly downwards self-reducible. Uniform $\mathcal{NC}^1$ is contained in $\oplus L$, so by Claims 2.16, 2.19 and 2.20, $DCMD$ is a randomized image of the decisional version of L_{S_5} that can be implemented by $\mathcal{NC}^0$ circuits. ■

3 Transformations: Reducing the Complexity of Decoding

In this section we show how to transform locally decodable and locally list-decodable codes with expensive decoding (say in $\mathcal{NC}^1$), into ones that have constant depth $\mathcal{AC}^0$ decoders.

3.1 Unique Local Decoding

We first prove Theorem 1.1 (see Section 1.2), showing how to transform a locally (and uniquely) decodable *binary* code that has an $\mathcal{NC}^1$ decoder to a locally decodable code (with almost the same parameters) that has an $\mathcal{AC}^0$ decoder. See the next section for transformations on (uniquely and list) locally decodable codes over general alphabets.

Proof Intuition for Theorem 1.1. This application of our general approach is the simplest one. The idea is that the new encoding of $x \in \{0,1\}^M$ has two (equally long) parts appended together. The first part is the original encoding of x and the second part is the truth-table of the randomized image I of the $\mathcal{NC}^1$-complete language L (given by Lemma 2.23) on instances of length $\log N$. Given a word that is close enough to a codeword in the new code, we know that both parts are close to what they should be. We then simulate in $\mathcal{AC}^0$ the original $\mathcal{NC}^1$ decoder (on the first half of the received word) as follows: every time we need to compute something (in $\mathcal{NC}^1$) that we cannot do in $\mathcal{AC}^0$, we reduce this computation to an instance y of the language L. We then use the random instance reduction from L to I (computable in $\mathcal{NC}^0$) to compute this y by querying the truth-table of I (i.e. the second half of the received word) at a random location. Since the second half is a string that is close to the truth-table of I, with relatively high probability we compute $L(y)$ correctly. We can increase the success probability (in $\mathcal{AC}^0$) by repeating many times in parallel and taking the approximate majority of the answers (this can be done in $\mathcal{AC}^0$ by [AB84, Ajt93]). Thus with high probability, the new $\mathcal{AC}^0$ decoder decodes the first half exactly as the old $\mathcal{NC}^1$ decoder does.

A subtle issue that we need to deal with is the fact that the I-instances we reduce to need to be of length exactly $\log N$ (so that the two parts of the code are equally long). To that end we use the strong downwards self reducibility property of L (see Definition 2.21 and Lemma 2.23) to adjust the lengths of the instances we work with. ■

Proof. (of Theorem 1.1) Let L be the $\mathcal{NC}^1$-complete language from Lemma 2.23, and let I be its randomized image. Let $x \in \{0,1\}^M$. We define:

$$C'(x)_i = C(x)_i \text{ if } 1 \leq i \leq N$$

and

$$C'(x)_i = I(i-N) \text{ if } N < i \leq 2N$$

That is, half of the codeword is the original codeword and the other half is the truth-table of I for input length $t = \log N$ (we assume w.l.o.g. that N is a power of 2). The approach to decoding C' is to use the random instance reduction from L to I to correct the I-half of $C'(x)$, and then to use the corrected I-half to help in decoding the half of $C'(x)$ corresponding to $C(x)$. Details follow.

We explain how to "compile" the $\mathcal{NC}^1$ decoder D for C, to create the $\mathcal{AC}^0$ decoder D' for C' that has the same output as D (w.h.p.). The key component in D' is a probabilistic $\mathcal{AC}^0$ procedure A, that tries to predict whether a given instance is in L or not, by querying the (possibly corrupted)

codeword y. Let $(\mathcal{R}, \mathcal{E})$ be the random instance reduction from L to I. Recall from Definition 2.9, that $\mathcal{R}$ is the randomizer that maps L-instances to random I-instances, and $\mathcal{E}$ is the evaluator that computes the membership status of an L-instance given the membership status of the random I-instance it was mapped to (by $\mathcal{R}$). Let $t' = t^\alpha$ (for some constant $\alpha > 0$) be such that $\mathcal{R}$ maps instances of length t' to instances of length t. We now describe the algorithm A:

Input : $a \in \{0,1\}^{t'}$

Output : A prediction for $L(a)$.

The algorithm:

1. Run $\mathcal{R}$ on a, $d = \Theta(\log N)$ times in parallel and with independent random coins to obtain $(c_1, \tau_1), \ldots, (c_d, \tau_d)$, where $c_j \in \{0,1\}^t$ and $\tau_j \in \{0,1\}$.
2. Query (in parallel) y at locations $c_1 + N, \ldots, c_d + N$ (where we view c_j as a binary representation of an instance in $[N]$) to obtain the bits $b_1, \ldots, b_d$. For each $1 \leq j \leq d$, let $w_j = \mathcal{E}(b_j, \tau_j)$.
3. Run the $\mathcal{AC}^0$ procedure from Lemma 2.6 that computes the approximate majority on $w_1, \ldots, w_d$ and output the outcome of the procedure.

Claim 3.1. *If y is at distance at most $\delta/2$ from a codeword in C' (where $\delta < 1/4$), then for every $a \in \{0,1\}^{t'}$, the probability that A predicts correctly the membership of a in L is at least $1 - 1/N$.*

Proof. Since y is at distance at most $\delta/2$ from a codeword in C', then if we consider only the I-half of y, it must be correct on at least a $1 - \delta > 3/4$ of the entries. By the properties of the random instance reduction, $c_1, \ldots, c_d$ are uniformly and independently distributed in the I-half of y. So for each $j \in [d]$, $\Pr[b_j = I(c_j)] \geq 3/4$. When $b_j = I(c_j)$, $w_j = L(a)$ (again by the properties of random instance reduction). By Chernoff bound, with probability at least $1/N$ at least $2/3$ of the w_j's hold the correct prediction for $L(a)$. And then the approximate majority procedure gives the correct answer. ■

Before describing D', let us consider the computation of D (given oracle access to a word $y' \in \{0,1\}^N$). Let $s = \operatorname{poly}\log(N)$ be the size of D as an $\mathcal{NC}^1$ circuit. We can divide D's computation as follows: first in parallel it computes the (bits of the) positions it is going to query y', and then given the values of y' in these locations it computes the output bit. Thus the execution of D amounts to running at most s different $\mathcal{NC}^1$ circuits, each of size at most s. Let $e_1, \ldots, e_k$ be the $\mathcal{NC}^1$ computations that compute the query locations, and e_{k+1} computes the output bit. By the fact that L is $\mathcal{NC}^1$-complete under $\mathcal{NC}^0$ reductions and it is strongly downwards self-reducible, we can solve every e_i by a constant depth circuit making queries to instances of L of length $(\log N)^\alpha = t'$.

We are now ready to describe D'. It is given randomness for the circuit D (and additional auxiliary randomness), as well as oracle access to a word $y \in \{0,1\}^{2N}$. It then simulates the execution of D when it is given oracle access to the first half of y (which we should think about as the original received word). As explained above, this computation can be divided to the $\mathcal{NC}^1$ computations $e_1, \ldots, e_{k+1}$. In parallel for each e_i $(1 \leq i \leq k)$ D' runs the constant-depth circuit that computes e_i by making queries to instances of L of length t'. This circuit makes $\operatorname{poly}\log(N)$

queries to L on instances of length t'. On each one of these queries, run the procedure A and use its answer as the oracle answer in the strong downwards self-reducibility circuit. If A does not make mistakes, then D' holds at this stage the query positions that D would have made to the original word in $\{0,1\}^N$. So D' makes the queries in the same locations D makes them (in the first half of y). After making all these queries, D' holds the answers that D would have read from the original word. Given these values and the randomness, D' can compute e_{k+1} in the same way it computed $e_1, \ldots, e_k$. If A does not make a mistake, then D' holds at this stage the output of D which we also define to be the output of D'.

We conclude that D' does exactly the same as D as long as A does not make any mistake during the execution. By claim 3.1, A makes a mistake with probability at most $1/N$. Since D' runs A at most $\operatorname{poly}\log(N)$ times, by the union bound, with probability almost 1, D' outputs the same output as D. Now, since y is at most $\delta/2$-far from a codeword in C', its first half is at most δ-far from a codeword in C. Therefore with high probability D, and hence D', retrieves the correct value of the message in the given location. By construction D' is an $\mathcal{AC}^0$ circuit of size $\operatorname{poly}\log(N)$. ■

3.2 Local List Decoding

In this section we prove Theorem 1.2, showing how to transform a locally list-decodable code that has an $\mathcal{NC}^1$ decoder to a locally list-decodable code that has an $\mathcal{AC}^0$ decoder. In particular, this theorem also implies a similar transformation for (uniquely) locally decodable codes with non-binary alphabets.

Proof Intuition for Theorem 1.2. The approach that we used to prove Theorem 1.1 cannot be used to recover from distance more than 1/4 (even if we start with a code that can be list-decoded from a large distance), because we need the truth-table part of the word to be more than 1/2-close to the respective half in the codeword. One thing we can do to improve the distance that we can recover from, at the price of doubling the alphabet size, is to append the truth-table *component-wise*. That is, we append to each bit of the original codeword an entry of the truth-table (recall that they are of the same length). This allows us to list-decode from distance $1/2 - 1/\log\log(N)$ (assuming the original code has a locally list-decoder in $\mathcal{NC}^1$), and the alphabet size is 4 (two bits per symbol). However, to recover from distance more than 1/2 requires a different technique. The idea now is to append (again, componentwise) the *direct-product* of I's truth table (I is, as before, the randomized image of the $\mathcal{NC}^1$-complete language L from Lemma 2.23). That is, every symbol in the new encoding contains a symbol from the old encoding concatenated with the binary string $I(i_1), \ldots, I(i_s)$, where $(i_1, \ldots, i_s)$ is a tuple of binary strings of some length that is determined in the proof. For every possible tuple we will have a different entry in the codeword.

As in the proof of Theorem 1.1, the new decoder simulates the old decoder. When it needs to compute some $\mathcal{NC}^1$ computation, it creates a uniformly distributed s-tuple where in each entry it either (with probability 1/2) puts a random instance for which it knows its correct membership status in I (this can be done by using the solved instance generator for I, see Definition 2.8), or (with probability 1/2) puts a random instance from which it can conclude the value of the $\mathcal{NC}^1$ computation given the correct membership status of that instance in I (this can be done using the random instance reduction from L to I and the fact that L is $\mathcal{NC}^1$-complete). The new decoder now reads from the corresponding location in the received word the (possibly corrupted) membership status of every instance in the s-tuple. It then checks whether on the entries for which it knows the correct answer, the received word is correct, and that all the other (randomized) instances evaluate

back to the same answer. If not, it declares this location in the received word to be corrupted. Otherwise it is a good indication that the symbol is not corrupted. The decoder assumes that the values it reads from it are all correct and infers from them the correct value of the $\mathcal{NC}^1$ computation. Given this procedure we can continue the simulation as in the proof of Theorem 1.1.

A subtle issue is the fact that the length of the original codeword and the length of the extra information we append to it (i.e. the number of s-tuples) are not necessarily the same. To solve this, we write many copies of the original codeword in the new one, so that the repeated original codeword is of the same length as the number of s-tuples. We proceed with the proof. ■

Proof. (Of Theorem 1.2) We start by describing the code. Let $s = s(N, \varepsilon)$ and $t = t(N, \varepsilon)$ be two integers that will be determined later. Consider all the s-tuples $(i_1, \ldots, i_s)$ where $i_j \in \{0,1\}^t$ (for every $1 \leq j \leq s$). We can assume, and will indeed make sure by our choice of s and t, that $2^{st}/N$ is an integer (and we assume without loss of generality that N is a power of 2). Let L be the $\mathcal{NC}^1$-complete language from Lemma 2.23 and let I be its randomized image. Let $t' = \text{poly}(t)$ be such that the random instance reduction from L to I on L-instances of length t' generates I-instances of length t.

Let $x \in \Sigma^M$, and let us define the codeword $C'(x)$. Every entry in this codeword is indexed by a different s-tuple as above. Thus the length N' of $C'(x)$ is 2^{st}. Let F be a (natural) bijection from s-tuples to $[2^{st}]$. Let $(i_1, \ldots, i_s)$ be an s-tuple and $k = (F(i_1, \ldots, i_s) \mod N)$. The entry in $C'(x)$ indexed by $(i_1, \ldots, i_s)$ is defined to be:

$$C'(x)_{i_1,\ldots,i_s} = C(X)_k \circ (I(i_1), \ldots, I(i_s))$$

where $\circ$ denotes concatenation of strings. That is, every entry contains two parts, one is an entry in the original encoding of x, and one is the s-bit characteristic vector of $i_1, \ldots, i_s$ as instances of the language I. The original encoding is repeated $2^{st}/N$ times in the new encoding. It is easy to see that if C is explicit then so is C'. Also note that every symbol in Γ is composed of a symbol from Σ and s bits. Thus $|\Gamma| = |\Sigma| \cdot 2^s$.

We now describe the decoder D' for C'. More specifically, we describe how to "compile" the $\mathcal{NC}^1$ decoder D for C, to create the $\mathcal{AC}^0$ decoder D' that has the same output as D. Namely, it outputs the $\mathcal{NC}^1$ circuits $M_1, \ldots, M_\ell$. Then each of the M_j's needs to be "compiled" to an $\mathcal{AC}^0$ circuit, but this can be done (in $\mathcal{AC}^0$) in the same way we transform D to D'.

The key component in D' is a probabilistic $\mathcal{AC}^0$ procedure that tries to learn whether a given instance is in L or not, by querying the (possibly corrupted) word y. We start by describing a procedure A_1 that makes one query to the codeword and has a very low probability of success, we later show how to amplify its success probability. A_1 either outputs a prediction regarding the membership of the given instance in L or declares the location in which it queries the codeword to be corrupted:

Input : $a \in \{0,1\}^{t'}$

Output : A prediction for $L(a)$ or "corrupted".

The algorithm:

1. For each $i \in [s]$ (in parallel and with independent random coins), do the following: Choose uniformly $v_i \in_R \{0,1\}$. If $v_i = 0$, run the $\mathcal{NC}^0$ solved instance generator for I

on input length t, to obtain a pair (c_i, y_i) where $y_i \in \{0,1\}$ and $c_i \in \{0,1\}^t$. If $v_i = 1$, run on a the $\mathcal{NC}^0$ random instance reduction from L to I, to obtain a pair (c_i, τ_i) where $\tau_i \in \{0,1\}$ and $c_i \in \{0,1\}^t$.

2. Query the codeword y at the position indexed by the s-tuple $(c_1, \ldots, c_s)$, to retrieve the symbol $\sigma \circ (b_1, \ldots, b_s)$ (where $\sigma \in \Sigma$ and $b_j \in \{0,1\}$).
3. For every $i \in [s]$ for which $v_i = 0$, check that $b_i = y_i$. If this equality does not hold (for at least one of these i's) declare the position indexed by $(c_1, \ldots, c_s)$ to be corrupted and abort.
4. For every $i \in [s]$ for which $v_i = 1$, compute the value $w_i = \mathcal{E}(b_i, \tau_i)$ (recall that $\mathcal{E}$ is the evaluator in the random instance reduction from L to I), and check that for all these i's the w_i's are equal. If they are, set $w = w_i$. Otherwise, declare the position indexed by $(c_1, \ldots, c_s)$ to be corrupted and abort.
5. Output w.

It is easy to verify that A_1 is in $\mathcal{AC}^0$. We now give bounds on the probabilities that it gives correct and wrong predictions.

Claim 3.2. *If y is ε-close to a codeword in C', then for every $a \in \{0,1\}^{t'}$, the probability that A_1 predicts correctly the membership of a in L is at least $\varepsilon - 2^{-s}$.*

Proof. By the definitions of solved instance generator and random instance reduction, $c_1, \ldots, c_s$ are independently and uniformly distributed. Thus we query y at a random location (uniformly distributed). Since y is ε-close to a codeword, with probability at least ε we query a non-corrupted location. When this happens, we get the correct prediction as long as not all the v_i's are 0 (which happens with probability 2^{-s}). The reason is that $b_1, \ldots, b_s$ are the true (0/1) membership values of the instances $c_1, \ldots, , c_s$ in I. By the definition of solved instance generator, this implies that with probability 1, for every i for which $v_i = 0$, $b_i = y_i$. And by the definition of random instance reduction, for every i for which $v_i = 1$, $w_i = L(a)$. ∎

Claim 3.3. *For every $a \in \{0,1\}^{t'}$ (and every y), the probability that A_1 gives the wrong prediction regarding the membership of a in L (i.e. says that it is in L when it is not or vice versa) is at most 2^{-s}.*

Proof. By the definitions of solved instance generator and random instance reduction, $c_1, \ldots, c_s$ are independently and uniformly distributed, and they are independent of the v_i's. Now suppose A_1 does not declare location $(c_1, \ldots, c_s)$ to be corrupted while giving a wrong prediction regarding the membership of a in L. This can only happen if $b_i = y_i = I(c_i)$ for every i for which $v_i = 0$ (by the definition of solved instance generator), and $b_i \neq I(c_i)$ for every i for which $v_i = 1$ (for otherwise, by the definition of random instance reduction, w_i will be equal to $L(a)$ for at least one of these i's). In other words, location $(c_1, \ldots, c_s)$ must be corrupted in such a way that exactly the bits indexed by i's for which $v_i = 1$ are flipped while the rest stay intact. Or, equivalently, every corruption pattern has exactly one choice of v_i's that will cause A_1 to give the wrong prediction (and not to detect the corruption). Since the v_i's are independent of $(c_1, \ldots, c_s)$, we conclude that the probability that this event happens is 2^{-s}. ∎

We now amplify the success probability of A_1. Consider the following $\mathcal{AC}^0$ procedure A_2:

Input : $a \in \{0,1\}^{t'}$

Output : A prediction for $L(a)$.

The algorithm:

1. Run d/ε times in parallel and with independent random coins the algorithm A_1 on input a (d will be determined later).
2. Look at the string of length d/ε over $\{0,1,*\}$ of the answers A_1 gives in the previous step (where $*$ stands for "corrupted"). Run the procedure from Lemma 2.5 on that string, to estimate the number of 0's and 1's in the string. I.e. first change every $*$ to 1 and estimate the number of 0's and then change every $*$ to 0 and estimate the number of 1's.
3. Output the value that gets the higher estimate.

Next we want to prove that A_2 gives the correct prediction with very high probability. This is true if we can ensure that (with high probability) the ratio between right and wrong predictions of A_1 is large (and then the procedure from Lemma 2.5 gives the right prediction). To that end we set s to be $\log(1/\varepsilon)+2$, and prove:

Claim 3.4. *If y is ε-close to a codeword in C', then for every $a \in \{0,1\}^{t'}$, the probability that A_2 predicts correctly the membership of a in L is at least $1-2^{-\Omega(d)}$.*

Proof. Let z be the string of length d/ε over $\{0,1,*\}$ of A_1's answers. Assume w.l.o.g. that $a \in L$ (otherwise replace 1's with 0's in the argument below). By Claims 3.2, 3.3 and our choice of s, the expected number of 1's in z is at least $(d/\varepsilon)(\varepsilon - 2^{-s}) = 3d/4$, and the expected number of 0's is at most $(d/\varepsilon)2^{-s} = d/4$. By Chenoff bound, with probability at least $2^{-\Omega(d)}$, the actual number of 1's will be at least $2d/3$ while the number of 0's at most $d/3$. The $\mathcal{AC}^0$ procedure from Lemma 2.5 estimates the number of 1's and 0's upto 0.01 (multiplicative) precision. This is enough for A_2 to conclude correctly that $a \in L$. ∎

Before we describe D', let us consider the computation of D (given oracle access to a word $y \in \Sigma^N$). Let $w = \text{poly}(\log(N)/\varepsilon)$ be the size of D as an $\mathcal{NC}^1$ circuit. As in the proof of Theorem 1.1, we can divide D's computation to the $\mathcal{NC}^1$ computations, $e_1,\ldots,e_K$, that compute the query locations. And the $\mathcal{NC}^1$ computations, $e_{K+1},\ldots,e_w$, that compute the output bits (unlike the proof of Theorem 1.1, now D outputs many bits which are the descriptions of the M_j's). Each e_i is of size at most w. By the fact that L is $\mathcal{NC}^1$-complete and it is strongly downwards self-reducible, we can solve every e_i by a constant depth circuit making queries to instances of L of length $(\log(N)/\varepsilon)^\alpha$ for an arbitrarily small constant $\alpha > 0$. We set t' to be $(\log(N)/\varepsilon)^\alpha$, and set α such that $t = (\log(N)/\varepsilon)^\delta(\log(1/\varepsilon)+2)^{-1}$ where δ is the arbitrarily small constant from the statement of the theorem. Recall that t and t' are polynomially related, so there is a setting of the parameters that achieve this. Also in our choice of parameters, we make sure that $2^{st}/N$ is an integer (we have enough freedom in our choice of t to do this).

We are now ready to describe D'. It is given randomness for the circuit D (and additional auxiliary randomness). Then in parallel for each e_i ($1 \le i \le K$) it runs the constant-depth circuit that computes L at length w by querying on instances of length t' (via strong downwards self-reducibility). This circuit makes $\text{poly}(\log(N)/\varepsilon)$ queries to L on smaller instances. On each one of these queries, run the procedure A_2 and use its answer as the oracle answer. If A_2 does not

make mistakes, then D' holds at this stage the query positions that D would have made to the original codeword in Σ^N. For each such position, D' makes the query in a random copy of the original codeword. That is, it chooses $p \in_R [2^{st}/N]$ and queries y in the position p plus the address computed for the original codeword. After making all these queries, D' holds the list of Σ-symbols that D would have read from the original codeword (At this stage D' ignores the s bits appended to each Σ-symbol). Given these values and the randomness, D' can compute $e_{K+1}, \ldots, e_t$ in the same way it computed $e_1, \ldots, e_K$. If A_2 does not make mistakes, then D' holds at this stage the output of D which we also define to be the output of D'.

We conclude that D' does exactly the same as D as long as A_2 does not make any mistake during the execution. The number of times we invoke A_2 is $\ell = \text{poly}(\log(N)/\varepsilon)$. We set d (the parameter from the description of A_2) to be $\Theta(\log(\log(N)/\varepsilon))$ with a leading constant that ensures that the probability (specified in Claim 3.3) that A_2 makes an error is at most $\ell^{-1}/100$. So the probability that D' fails to decode correctly is bounded by the error probability of D (at most $1/4$), plus the probability that A_2 makes an error (at most $1/100$ by union bound). By running D' twice with independent random coins we can drive the error down to below $1/4$. If the code was uniquely decodable it remains uniquely decodable. If it was locally list-decodable (with $\ell > 1$) then this incurs a price of doubling the list size. It is easy to verify that D' can be implemented by $\mathcal{AC}^0$ circuits of size $\text{poly}(\log(N)/\varepsilon)$.

Going back to the parameters of the code we see that the length N' of a codeword is

$$2^{st} = 2^{\log(1/\varepsilon+2)(\log(N)/\varepsilon)^{\delta}(\log(1/\varepsilon)+2)^{-1}} = 2^{(\log(N)/\varepsilon)^{\delta}}$$

and the size of the alphabet Γ is $|\Sigma| \cdot 2^s = |\Sigma| \cdot O(1/\varepsilon)$. ■

4 Explicit Constructions

In this section we apply our general theorems from the previous sections to construct codes with $\mathcal{AC}^0$ local-decoders. We do that by first constructing codes with $\mathcal{NC}^1$ decoders and then applying the general transformations to them. Previous locally-decodable codes with the parameters that we need are not known to be in $\mathcal{NC}^1$ (in particular decoding the code given in [STV01] involves solving a system of linear equations). We therefore construct new explicit codes with $\mathcal{NC}^1$ decoders, and then apply our transformations to them.

4.1 A (Uniquely) Locally Decodable Binary Code

We prove theorem 1.3, constructing a binary code that can be (uniquely) locally-decoded from a constant distance.

Proof of Theorem 1.3. In order to apply Theorem 1.1 we need to show an explicit code that can be non-adaptively locally decoded from distance $2/25$ by $\mathcal{NC}^1$ circuits of polylogarithmic size. To the best of our knowledge the only known locally-decodable code with the parameters that we need is the so called low-degree extension code concatenated with Hadamard [STV01]. Unfortunately we do not know how to implement the decoder for this code in $\mathcal{NC}^1$. The reason is that it involves decoding the Reed-Solomon code which is done via solving a system of linear equations. We therefore use a combination of known constructions and techniques to construct a new code that does have an $\mathcal{NC}^1$ local decoder (and which avoids decoding the Reed-Solomon code). Our construction has three stages.

Stage 1 - Decoding the low-degree extension from small distances. The first stage shows how to decode the low-degree extension code from a very small (sub-constant) relative distance.

Given a string $x \in \{0,1\}^M$ we associate with it a multi-variate polynomial over a finite field $\mathbb{F}$ as follows: fix $\mathbb{F}$ to be a field of cardinality $(\log M)^2$. Fix a subset H of the field of cardinality $\log M$. Let m be $(\log M)/(\log |H|)$. Let $b : [M] \to H^m$ be an injective map. To encode x, we find the unique m-variate polynomial $\bar{x} : \mathbb{F}^m \to \mathbb{F}$ of degree at most $|H| - 1$ in each of the m variables, such that for every $i \in [M]$, $\bar{x}(b(i)) = x_i$. Such a polynomial can be efficiently found by means of (multi-variate) interpolation. The encoding of x is the evaluation of $\bar{x}$ on every point in $\mathbb{F}^m$. Let us call this code C_1 and it maps $\{0,1\}^M$ to $\mathbb{F}^{N_1}$, where $N_1 = |\mathbb{F}|^m$. Note that $N_1 = M^2$, this is because,

$$\log N_1 = m \log |\mathbb{F}| = \frac{(\log M)(\log |\mathbb{F}|)}{\log |H|} = 2 \log M$$

We now explain how to locally-decode C_1 from distance $\delta_1 = 1/(10|\mathbb{F}|) = 1/(10(\log M)^2)$. We are given a string $y \in \mathbb{F}^n$ such that there exist a string $x \in \{0,1\}^M$ that satisfies $\Delta(C_1(x), y) \leq 1/(10|\mathbb{F}|)$, and an index $i \in [M]$. We associate i (via b) with a point in $H^m \subseteq \mathbb{F}^m$ on which we want to evaluate the polynomial $\bar{x}$. We choose a random line in $\mathbb{F}^m$ that passes through $b(i)$ (i.e. for $h \in_R \mathbb{F}^m$, we look at the line $\{b(i) + hz : z \in \mathbb{F}\}$), and we read from y the $\mathbb{F}$-values associated with the locations of all the $|\mathbb{F}|$ points on that line. By pairwise independence, every point on the line in itself is uniformly distributed (independent of $b(i)$), so by the union bound, with probability at least $9/10$, all the values that we read agree with $\bar{x}$ restricted to the line. Note that the restriction to the line is a univariate polynomial over $\mathbb{F}$ of degree

$$m(|H| - 1) \leq (\log M)^2 / \log \log M < |\mathbb{F}|$$

So by interpolation we can retrieve $\bar{x}(b(i))$. By using Lagrange's formula and [HV06], this non-adaptive decoder can be implemented by $\mathcal{NC}^1$ circuits of size $\text{poly}|\mathbb{F}| = \text{poly}(\log M)$.

Stage 2 - Boosting the distance with expanders. We now show how to obtain from C_1 a code with large alphabet that can be decoded from a constant distance. To that end we use the expanders based technique of [ABN+92]. Specifically, let $G = (L, R, E)$ be a d-regular bipartite graph where $|L| = |R| = N_1$, and G has the property that for every $B \subseteq R$ with $|B| \leq 2N_1/5$, there are at most $\delta_1 N_1$ vertices $v \in L$, such $|\Gamma(v) \cap B| \geq d/2$ (where $\Gamma(v)$ is the set of neighbors of v).

Claim 4.1. *There exist a family of graphs G with the specified parameters, where $d = \text{poly}(1/\delta_1)$, and given a name of a vertex v in G, the list of its d neighbors can be computed by $\mathcal{NC}^1$ circuits of size* $\text{poly}(\log N_1, 1/\delta_1)$.

Proof. Let H be the regular constant-degree Gabber-Galil expander on N_1 nodes [GG81]. We take G to be the bipartite $c\log(1/\delta_1)$-th power of H for some constant c that will be chosen later: specifically, we join $x \in L$ and $y \in R$ by an edge for each walk of length $c\log(1/\delta_1)$ from x to y in H. Since H has constant degree, the degree of G is $d = O(1)^{c\log(1/\delta_1)} = (1/\delta_1)^{O(c)}$. Moreover, it has been shown [GG81] that H has (normalized) second-largest eigenvalue $\lambda = 1 - \Omega(1)$ and it is well known that $\lambda(H^k) = \lambda(H)^k$ for regular undirected graphs H. Therefore, our graph G has $\lambda(G) = (1 - \Omega(1))^{c\log(1/\delta_1)} = \delta_1^{\Theta(c)}$.

We now prove that G has the required decoding property. Fix a subset $B \subseteq R$ with $|B| \leq 2N_1/5$ and let A be the set of vertices $v \in L$, such that $|\Gamma(v) \cap B| \geq d/2$ (where $\Gamma(v)$ is the set of neighbors of v). We wish to show that $|A| \leq \delta_1 N_1$. By the expander mixing lemma (cf. [AS00]), we have:

$$\left| E(A,B) - d \cdot \frac{|A| \cdot |B|}{N_1} \right| \leq d\lambda\sqrt{|A| \cdot |B|}.$$

By the definition of A and B, $E(A,B) \geq \frac{d}{2} \cdot |A|$, and so we have:

$$\frac{d}{2} \cdot |A| - d \cdot \frac{|A| \cdot |B|}{N_1} \leq d\lambda\sqrt{|A| \cdot |B|},$$

which implies that $|A| \leq \lambda^2 \cdot |B|/(1/2 - |B|/N_1)^2 \leq 40 \cdot \lambda^2 \cdot N_1$ (where the last inequality follows from the assumption that $|B| \leq 2N_1/5$). Since $\lambda = \delta_1^{\Theta(c)}$, we have $|A| \leq O(\delta_1^{\Theta(c)})N_1 \leq \delta_1 N_1$ by an appropriate choice of the constant c.

Furthermore, it has been shown [GV04] that walks of length ℓ on the N-node Gabber-Galil expander can be computed by $\mathcal{AC}^0$ circuits of size $\text{poly}(\log N, 2^\ell)$, so the neighbors of a given node in G can actually be computed by $\mathcal{AC}^0$ circuits of size $\text{poly}(\log N_1, 1/\delta_1)$ (and therefore, by $\mathcal{NC}^1$ circuits of the same size). ∎

We define the following code $C_2 : \mathbb{F}^{N_1} \to (\mathbb{F}^d)^{N_1}$:

$$C_2(x)_i = x_{\Gamma_1(i)}, x_{\Gamma_2(i)}, \ldots, x_{\Gamma_d(i)}$$

where $\Gamma_j(i)$ is the j'th neighbor of vertex $i \in R$.

We present a local-decoder D_2 for C_2 that given a string $y \in (\mathbb{F}^d)^{N_1}$ for which there exist $x \in \mathbb{F}^{N_1}$ that satisfies $\Delta(y, C_2(x)) \leq 2N_1/5$, the decoder computes every entry in x correctly except

for a δ_1 fraction of the entries. The decoder is as follows: Given oracle access to y as above and an index $i \in N_1$, we think of i as a left vertex in G. The decoder computes the list of right neighbors $j_1, \ldots, j_d$ of i. It then query y in positions $j_1, \ldots, j_d$, to retrieve d d-tuples of $\mathbb{F}$-symbols. Each such tuple contains a prediction of x_i. That is, if i is the k'th neighbor of j_1 then the k'th entry in the d-tuple that we read from location j_1 in y is x_i (supposedly, if j is not a corrupted location). The decoder takes x_i to be the prediction that appears more often than others in the d queries (in fact taking the majority is enough). Note that D_2 can be implemented by $\mathcal{NC}^1$ circuits of size $\text{poly}(\log N_1, d) = \text{poly}(\log N_1, 1/\delta_1)$, this is because the neighbors in G can be computed in $\mathcal{NC}^1$ by Claim 4.1, and computing the plurality amounts to comparing in parallel all the d predictions and counting (in $\mathcal{NC}^1$) which one appears most often. We now prove correctness.

Claim 4.2. $\Pr_i[D_2^y(i) = x_i] \geq 1 - \delta_1$

Proof. Let $B \subseteq R$ be the set of right vertices that are associated with the corrupted locations in y. By the hypothesis, $|B| \leq 2N_1/5$. Let $A \subseteq L$ be the set of vertices $v \in L$ such that $|\Gamma(v) \cap B| > d/2$. By the properties of G, $|A| \leq \delta_1 N_1$. For every $i \notin A$, the majority of $y_{\Gamma_1(i)}, \ldots, y_{\Gamma_d(i)}$ are not corrupted, and therefore will give the right prediction for x_i. ∎

Combining the codes C_1 and C_2, we get a code $C' : \{0,1\}^M \to (\mathbb{F}^d)_1^N$ that can be non-adaptively locally-decoded from a constant distance by $\mathcal{NC}^1$ circuits. On index $i \in [M]$, the decoder D' for C', first runs the decoder D_1 for C_1 to produce the query locations. These locations are then passed to the decoder D_2 for C_2. D_2 retrieves the values in the requested locations and passes them back to D_1, who then computes the i'th bit of the message. D' is in $\mathcal{NC}^1$ because D_1 and D_2 are (and all the queries are done in parallel). Informally, it corrects from distance $2/5$ because D_2 corrects all but δ_1 fraction of errors, and D_1 corrects the remaining δ_1.

Stage 3 - Back to binary code. The code C' can be locally-decoded from a constant distance, however the alphabet of the code is not binary but rather contains $|\mathbb{F}|^d = (\log M)^{\text{poly}\log M}$ symbols. To get back to a binary code we use the technique of concatenating codes. More precisely, note that every symbol in $\mathbb{F}^d$ has a description (by bits) of size $\text{poly}\log M$. We encode every symbol separately by a binary code of polynomial rate. Thus the size of the new codeword grows only by a polylogarithmic factor. The code that we are going to use is the low-degree extension code concatenated with Hadamard [STV01]. Recall that previously we said that we do not know of an $\mathcal{NC}^1$ decoder for this code, however now we need to encode and decode strings of exponentially smaller length (i.e. instead of length M, length $\text{poly}\log M$), and this can easily be done by $\mathcal{NC}^1$ circuits of size $\text{poly}\log M$.

Let $M' = d \log |\mathbb{F}|$, consider the binary code $C_3 : \{0,1\}^{M'} \to \{0,1\}^{N'}$ given in [STV01] (i.e. the low-degree extension code concatenated with Hadamard), that has $N' = \text{poly}(M')$ and can be decoded from distance $1/5$ (here we do not even need the code to be locally-decodable, i.e. the circuit of size $\text{poly}\log M$ can read the whole $(\text{poly}\log M)$-bit codeword).

Claim 4.3. *C_3 can be decoded from distance $1/5$ by $\mathcal{NC}^1$ circuits of size $\text{poly}\log(M)$.*

We define the concatenated code $C : \{0,1\}^M \to \{0,1\}^{N' \cdot N_1}$ as follows. Let $f : [N' \cdot N_1] \to [N'] \times [N_1]$ be a bijection. Let $i \in [N' \cdot N_1]$, and $f(i) = (j,k)$. We define,

$$C(x)_i = C_3(C'(x)_k)_j$$

Note that $N' = \text{poly}(d \log |\mathbb{F}|) = \text{poly} \log M$, and $N_1 = M^2$. So C maps M bits to poly(M) bits. We now describe the local decoder D for C. Given oracle access to a string y that is $2/25$ close to a codeword, and $i \in [M]$, we first run the decoder D' for C' to compute the query locations. For every such query, we read the N' bits that encodes (via C_3) the relevant symbol in the code C'. We run the decoder D_3 for C_3 to retrieve the symbol in the code C'. We then pass these symbols to D' who returns the i'th bit of the message. Clearly, D can be implemented by $\mathcal{NC}^1$ circuits of size $\text{poly} \log M$, because D' and D_3 can (and all the queries are done in parallel). We now prove correctness.

Claim 4.4. *D decodes C from distance $2/25$.*

Proof. Let $y \in \{0,1\}^{N' \cdot N_1}$ be a string such that there is $x \in \{0,1\}^M$ satisfying $\Delta(y, C(x)) \leq 2/25$. We think of the locations of bits in y as indexed by $(j,k) \in [N'] \times [N_1]$ (as in the definition of C). Let $B \subseteq [N_1]$ be such that for every $k \in B$,

$$\Pr_{j \in_R [N']}[y_{jk} \text{ is corrupted}] \geq 1/5$$

By Markov inequality,

$$\Pr_{k \in_R [N_1]}[k \in B] \leq 2/5$$

Since the decoder D_3 for C_3 decodes from distance $1/5$, it can decode correctly at least $3/5$ of the symbols (those that their indices are not in B) to obtain a string $y' \in (\mathbb{F}^d)^{N_1}$ satisfying $\Delta(C'(x), y') \leq 2/5$. The decoder D' for C' can decode from distance $2/5$, and therefore can retrieve (w.h.p.) every bit in the message x. ∎

Finally, we apply on the code C the transformation given in Theorem 1.1, to obtain an explicit binary code that maps M bits to poly(M) bits, and can be locally-decoded from distance $1/25$ by probabilistic $\mathcal{AC}^0$ circuits of size poly$(\log M)$. ∎

4.2 A List-Locally Decodable Non-Binary Code

Next we prove Theorem 1.4, showing how to explicitly construct a code with large alphabet that can be locally list decoded from very large distances by $\mathcal{AC}^0$ circuits. This code was originally presented in [GGH+07].

Proof of Theorem 1.4. As in the proof of Theorem 1.3, we first show a code with similar parameters that can be locally list-decoded by an $\mathcal{NC}^1$ decoder, and then we apply Theorem 1.2. Our code is given by applying the "direct product" construction of Impagliazzo et. al. [IJK06] on the code from Theorem 1.3. We start with some definitions. Recall the definition of an approximate locally list-decodable code [Tre03] (Definition 2.4). The following definition appears in [IJK06]:

Definition 4.5. [direct product codes] Let $C : \{0,1\}^M \to \{0,1\}^N$ be a code. The k-direct-product of C is the code $C^k : \{0,1\}^M \to (\{0,1\}^k)^{N^k}$, defined as follows (for $x \in \{0,1\}^M$ and $i_1, \ldots, i_k \in [N]$),

$$C^k(x)_{i_1,\ldots,i_k} = C(x)_{i_1}, \ldots, C(x)_{i_k}$$

So far, to keep the statements and proofs simple, we only considered non-adaptive decoders. In order to use [IJK06] we need to consider adaptive decoders.

Definition 4.6. We say that a decoder is k-adaptive (or has adaptivity k), if the queries that it makes to the received word can be partitioned into k sets (layers) where all the queries in layer i can be done simultaneously (given the answers to the queries from layers $j < i$).

It is not difficult to see that Theorems 1.1 and 1.2 can be generalized to decoders with constant adaptivity. This is done by applying the same arguments, from one layer to the next. Since the number of layers is constant, the new decoder is still in $\mathcal{AC}^0$.
The following is implicit in [IJK06].

Lemma 4.7. *Let $C : \{0,1\}^M \to \{0,1\}^N$ be an arbitrary code. Then for any $\varepsilon = \Omega(\mathrm{poly}(1/k))$: the k-direct-product of C is $(1/25)$-approximate locally and list-decodable from agreement ε with list size $\mathrm{poly}(1/\varepsilon)$ by constant-adaptivity probabilistic $\mathcal{NC}^1$ circuits of size $\mathrm{poly}(\log(N)/\varepsilon)$.*

Claim 4.8. *Let $C : \{0,1\}^M \to \{0,1\}^{\mathrm{poly}(M)}$ be the code from Theorem 1.3. For $\varepsilon > 0$, let $k = \mathrm{poly}(1/\varepsilon)$ satisfy the condition of Lemma 4.7. Then C^k, the k-direct-product of C, is an explicit code that is locally list-decodable with constant adaptivity by probabilistic $\mathcal{NC}^1$ circuits of size $\mathrm{poly}(\log(M)/\varepsilon)$ from agreement ε and with list size $\ell = \mathrm{poly}(1/\varepsilon)$.*

Proof. Let $x \in \{0,1\}^M$, and let $N = \mathrm{poly}(M)$ be the length of a codeword in C. Given oracle access to a string $y \in (\{0,1\}^k)^{N^k}$ satisfying $\Delta(y, C^k(x)) \leq 1 - \varepsilon$, the decoder D^k for C^k, runs the $(1/25)$-approximate local and list decoder for C^k. By Lemma 4.7, this produces a list, $M_1, \ldots, M_\ell$, of machines such that at least one of them, M_j, computes $C(x)$ correctly on at least a $24/25$ fraction of entries. We combine each M_i with the decoder for C. By Theorem 1.3, the latter can locally decode C from distance $1/25$. So together with M_j it computes x correctly on every entry. The decoder and the M_i's are in $\mathcal{NC}^1$ because so are the approximate decoder for C^k (by Lemma 4.7) and the decoder for C (Theorem 1.3). ■

Finally, we apply Theorem 1.2 (generalized to decoders with constant adaptivity) on C^k to obtain a code with the specified parameters that is locally list-decodable by probabilistic $\mathcal{AC}^0$ circuits of size $\mathrm{poly}(\log(M)/\varepsilon)$. ■

4.3 A Locally List-Decodable Binary Code

We now prove Theorem 1.5, giving a construction of locally list-decodable binary codes with efficient decoders. We obtain an essentially optimal (up to polynomial factors) construction, see Section 5 for the matching lower bounds.

Remark 4.9. *The construction of Theorem 1.5 only applies for $\varepsilon \geq 2^{-\Theta(\sqrt{\log M})}$. Thus we fall slightly short of covering the whole possible range (since one can hope to get such codes for $\varepsilon = 1/M^c$ for a small constant c). We note, however, that the range of ε which is most interesting for us is between $1/\mathit{poly}\log M$ and $1/\mathit{poly}\log\log M$ (see the discussion in the introduction) which we do cover. We also mention that if one insists on codes with $\varepsilon = 1/M^c$, then we can construct such codes with quasi-polynomial rate.*

To prove Theorem 1.5, we concatenate three codes. The first is the binary locally-decodable code constructed in Section 4.1 that can be uniquely decoded from a constant relative distance. I.e., the code of Theorem 1.3: $\{C_M : \{0,1\}^M \to \{0,1\}^{\mathrm{poly}(M)}\}_{M \in \mathbb{N}}$ that can be locally decoded (uniquely, i.e. with list size 1) from distance $1/25$ by probabilistic $\mathcal{AC}^0$circuits of size $\mathrm{poly}(\log M)$.

The second code that we need is a non-binary approximate locally-list-decodable code. Recall the definition of approximate locally list-decodable codes (Definition 2.4). We construct an approximate locally list-decodable code: a modification of the code of [IJKW08], that allows local decoding in $\mathcal{AC}^0$.

Theorem 4.10. *For every $\delta = O(1)$ and every $2^{-\Theta(\sqrt{\log M})} \leq \varepsilon = \varepsilon(M) < \delta$, there exists a δ-approximate $(\varepsilon, \mathrm{poly}(1/\varepsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \Gamma^{\mathrm{poly}(M)}\}_{M\in\mathbb{N}}$. Where $|\Gamma| = \mathrm{poly}(1/\varepsilon)$. The code has a local decoder that can be implemented by constant depth circuits of size $\mathrm{poly}(logM, 1/\varepsilon)$.*

Proof. The code of [IJKW08] is a concatenation of two approximate locally list-decodable codes. The "outer" code C_{out}, maps $\{0,1\}^M$ to $\Sigma^{\mathrm{poly}(M)}$, where $|\Sigma| = 2^{\mathrm{poly}(1/\varepsilon)}$. To reduce the alphabet size, each $\mathrm{poly}(1/\varepsilon)$-bit symbol of this code is then itself encoded by an "inner" code C_{in}, mapping $\{0,1\}^{\mathrm{poly}(1/\varepsilon)}$ to $\Gamma^{N'}$ where Γ is as stated in the theorem (of size $\mathrm{poly}(1/\varepsilon)$), and $N' = (\mathrm{poly}(1/\varepsilon))^{\log(1/\varepsilon)}$. We note that for the range of ε that we consider we get that $N' \leq \mathrm{poly}(M)$. Thus the concatenated code maps $\{0,1\}^M$ to $\Gamma^{\mathrm{poly}(M)}$, and by [IJKW08], its approximate list-decoding parameters are as stated.

In terms of complexity, the decoder for C_{in} can be implemented in $\mathcal{AC}^0$ (this is shown in [IJKW08]). We do not, however, know how to implement the decoder for C_{out} in $\mathcal{AC}^0$. Therefore, to get a code decodable in $\mathcal{AC}^0$, we modify their outer code (we abuse notation and still call the modified code C_{out}) and present an $\mathcal{AC}^0$ decoder for the modified code. The main reason we need to modify their code is that we don't know of a concise and unique representation of low-degree affine sub-spaces that can be computed and manipulated in $\mathcal{AC}^0$. We instead represent such subspace using some basis vectors and a shift vector (a concise, but not unique representation). This changes the code and allows decoding in $\mathcal{AC}^0$.

Modified Outer Code. Let $k = \log|\Sigma|$. We associate the set of message indices $[M]$ with a m-dimensional vector-space over a finite field $\mathcal{F}_q$, where $q^m = M$, and $q^8 = k = \log|\Sigma| = \mathrm{poly}(1/\varepsilon)$. The codeword is indexed by tuples of 9 vectors in $\mathcal{F}_q^m$, and thus each codeword is of length $O(|\mathcal{F}_q^m|^9) = O(M^9) = \mathrm{poly}(M)$.

Take $msg \in \{0,1\}^M$. Its encoding is defined as follows. Let $(\vec{v_1}, \ldots, \vec{v_8}, \vec{s})$ be an index into the codeword (i.e. $\vec{v_1}, \ldots, \vec{v_8}, \vec{s} \in \mathcal{F}_q^m$). Consider the affine subspace B spanned by $\vec{v_1}, \ldots, \vec{v_8}$ and shifted by $\vec{s}$

$$B = \{a_1\vec{v_1} + \cdots + a_8\vec{v_8} + \vec{s} : a_1, \ldots, a_8 \in \mathcal{F}_q\}$$

Recalling that entries in the message are associated with vectors in $\mathcal{F}_q^m$ (using some canonical ordering), we take the $(\vec{v_1}, \ldots, \vec{v_8}, \vec{s})$-th codeword entry to be all the message bits whose indices are in the affine subspace B (in some order). I.e.,

$$C_{out}(msg)[\vec{v_1}, \ldots, \vec{v_8}, \vec{s}] = \{msg[u] : u \in B\}$$

Each alphabet symbol's length is the size of the 8-dimensional subspace which is $q^8 = \mathrm{poly}(1/\varepsilon)$, and thus the alphabet size is exponential in $1/\varepsilon$, as claimed above. We think of each (9-vector) index in the codeword as an 8-dimensional affine subspace of $\mathcal{F}_q^m$, where the first 8 vectors in the index are basis vectors and the last one is a shift vector. First, note that each such affine subspace appears multiple times with different representations. Also note that not every index represents an 8-dimensional subspace (since the first 8 vectors may be linearly dependent). However, the next claim says that the number of indices that do not represent 8-dimensional subspaces is very small.

Claim 4.11. *Let $\vec{v_1}, \ldots, \vec{v_8}$ be chosen uniformly and independently from $\mathcal{F}_q^m$. The probability that they are linearly dependent is at most $\frac{q^8}{q^m} \leq 1/\sqrt{M}$.*

Proof. Examine the process of iteratively choosing 8 uniformly random vectors. After choosing the first $i \geq 0$ vectors, and assuming they are independent, they span a subspace of size q^i. The $i+1$-th vector is in the subspace they span only with probability $\frac{q^i}{q^m}$, and otherwise the $i+1$ vectors are linearly independent. Taking a Union Bound, the total probability the 8 vectors are dependent is at most $\frac{q^8}{q^m}$. ■

Note that in our range of parameters, $1/\sqrt{M} << \varepsilon$. So we can ignore the locations that are indexed by subspaces that are not 8-dimensional, this will not have a meaningful effect on the success probability of decoding or on Hamming distances between received words and codewords (since we only care about received words that have agreement at least ε with codewords).

We now describe the local decoder D for this code (viewed as a two-stage process as discussed in Remark 2.2). Given access to a string y that has Hamming distance at most ε from some codeword $C_{out}(msg)$, D produces $O(1/\varepsilon^2)$ probabilistic circuits such that at least one of them decodes at least $1-\delta$ fraction of the bits in msg (with high probability over the random coins of the circuit). Each one of the circuits in the list is generated independently as follows: D chooses uniformly at random 9 vectors $\vec{v_1}, \ldots, \vec{v_8}, \vec{s} \in \mathcal{F}_q^m$. By Claim 4.11, with high probability these vectors represent an 8-dimensional affine subspace B (otherwise consider this a failure). D then chooses a random 4-dimensional affine subspace $A \subseteq B$. Next, D reads the received word y in the entry indexed by $(\vec{v_1}, \ldots, \vec{v_8}, \vec{s})$. (For an uncorrupted location this should give the message values at all indices in the subspace B.) Let v be the values in this entry that are associated with the subspace A. D generates the circuit $C_{A,v}$ that does the following: on input $j \in [M]$, $C_{A,v}$ checks whether $j \in A$ (where we think of j as a vector in $\mathcal{F}_q^m$). If so, then $C_{A,v}$ outputs the bit in v that is associated with the vector j. Otherwise, $C_{A,v}$ repeatedly (in parallel) for $O(\frac{\log(1/\delta)}{\varepsilon})$ iterations does the following: chooses $(\vec{w_1}, \ldots, \vec{w_8}, \vec{t})$ that are randomly distributed under the constraint that they span an 8-dimensional affine subspace B, such that $A \cup \{j\} \subseteq B$. It then reads y at index $(\vec{w_1}, \ldots, \vec{w_8}, \vec{t})$, and compares the bits in v with the bits in this location associated with the elements of A. If for none of the iterations there is a full agreement between v and the bits associated with A, then $C_{A,v}$ outputs some error message. Otherwise, it takes the first iteration in which there is an agreement and outputs the bit in this location associated with j.

The proof that this decoder has the desired list-decoding properties follows exactly the proof of [IJKW08]. (Ignoring the tiny fraction of indices that are not 8-dimensional subspaces, our code is their code with the only difference that each bit in the codeword is repeated many times, as many as the number of representations of an 8-dimensional affine subspace.) We only need to verify that the complexity of the decoder is as stated, which amounts to verifying the following three easy claims. Note throughout that addition and multiplication over F_q can be done by $\mathcal{AC}^0$ circuits of size $\text{poly}(q) = \text{poly}(1/\varepsilon)$.

Claim 4.12. *Given an 8-dimensional affine subspace $B \subseteq \mathcal{F}_q^m$, represented by $(\vec{v_1}, \ldots, \vec{v_8}, \vec{s})$. A probabilistic $\mathcal{AC}^0$ circuit of size $\text{poly}(\log M, 1/\varepsilon)$ can sample a uniformly distributed 4-dimensional subspace $A \subseteq B$.*

Proof. Every 4-dimensional subspace in B is spanned by 4 linearly independent vectors, and all these sub-spaces are of equal size. Thus, to sample the subspace A we first sample 4 random vectors

$(\vec{w_1}, \ldots, \vec{w_4})$ in B, by taking random linear combinations of $\vec{v_1}, \ldots, \vec{v_8}$. With high probability (more than say $1/2$), these vectors will be linearly independent. This can be verified in $\mathcal{AC}^0$ by enumerating all $q^4 = \text{poly}(1/\varepsilon)$ linear combinations, and thus the success probability can be amplified (in parallel time) to $1 - 1/\text{poly}(M, 1/\varepsilon)$. This gives us a random basis for the 4-dimensional subspace A. Now, to choose the shift vector $\vec{t}$ we just choose another random vector in B (linearly independent or not) and obtain the random 4-dimensional subspace A spanned by $(\vec{w_1}, \ldots, \vec{w_4})$ and shifted by $\vec{t}$. All of these operations can be done by an $\mathcal{AC}^0$ circuit of size $\text{poly}(\log M, 1/\varepsilon)$. ■

Claim 4.13. *Given a 4-dimensional affine subspace $A \subseteq \mathcal{F}_q^m$, represented by $(\vec{v_1}, \ldots, \vec{v_4}, \vec{s})$ and a vector $\vec{j} \in \mathcal{F}_q^m$. An $\mathcal{AC}^0$ circuit of size $O(\log 1/\varepsilon)$ can check whether $\vec{j} \in A$.*

Proof. Again, this is easily done by enumerating all $q^4 = \text{poly}(1/\varepsilon)$ possible linear combinations of $(\vec{v_1}, \ldots, \vec{v_4})$, and checking whether any of them give the vector $\vec{j} - \vec{s}$. ■

Claim 4.14. *Given a 4-dimensional affine subspace $A \subseteq \mathcal{F}_q^m$, represented by $(\vec{v_1}, \ldots, \vec{v_4}, \vec{s})$ and a vector $\vec{j} \in \mathcal{F}_q^m \setminus A$, a probabilistic $\mathcal{AC}^0$ circuit of size $\text{poly}(\log M, 1/\varepsilon)$ can sample vectors $(\vec{w_1}, \ldots, \vec{w_8}, \vec{t})$ that are uniform under the condition that they span an 8-dimensional subspace B, such that $A \cup \{\vec{j}\} \subseteq B$.*

Proof. First, we take a fifth basis vector, which is $\vec{j} - \vec{s}$, to get the (unique) 5-dimensional subspace A' that contains A and $\{\vec{j}\}$. Now, we want to choose a random 8-dimensional subspace that contains A', and to do this we choose 3 more uniformly random vectors that are independent of $(\vec{v_1}, \ldots, \vec{v_4}, \vec{j} - \vec{s})$ (as above, this can be done in $\mathcal{AC}^0$).

We now have 8 basis vector and the shift vector $\vec{s}$. These define the 8-dimensional subspace B. Note that this representation of B depends on the representation of A (e.g. in particular the shift vector is still $\vec{s}$), while we need to choose an independent representation of B. To solve this, we "randomize" the basis vectors by choosing a new basis consisting of 8 random linearly independent linear combinations of the basis vectors (this randomizes the basis vectors but doesn't change the subspace), and then choosing a random vector in the subspace as the new shift. All of this is easily done in $\mathcal{AC}^0$, and we indeed obtain a random representation of a random subspace containing A'. ■

■

The third code in our construction is the well known Hadamard code with its local list-decoder given by Goldreich and Levin [GL89].

Theorem 4.15. *For every $0 < \varepsilon(m) < 1/2$, The Hadamard code, $Had : \{0,1\}^m \to \{0,1\}^{2^m}$ is a $(1/2 - \varepsilon, \frac{1}{\varepsilon^2})$-local-list-decodable-code. The decoder can be implemented by an $\mathcal{AC}^0$ circuit of size $\text{poly}(m, 1/\varepsilon)$ that uses majority gates of fan-in $\Theta(1/\varepsilon)$.*

We can now put everything together and prove Theorem 1.5.

Proof of Theorem 1.5. Fix $\varepsilon(M) = \varepsilon$ in the specified range. Our code C is a combination of the codes in Theorem 1.4 (we denote it here by C_1), the code in Theorem 4.10 (we denote it here by C_2) with $\varepsilon_2 = \varepsilon^3/2$ and $\delta = 1/25$, and the code in Theorem 4.15 (C_3), with $\varepsilon_3 = \varepsilon/2$. Given a message $x \in \{0,1\}^M$, we first encode it using C_1 to obtain a binary string x' (of length $N_1 = \text{poly}(M)$).

We then encode x' using C_2 to obtain a string of length $N_2 = \text{poly}(M)$ over the alphabet Γ (of size $\text{poly}(1/\varepsilon)$). We then concatenate this code (i.e. encode every symbol of it) with C_3. Let $k = \log(|\Gamma|)$. The length N of the final code is N_2 multiplied by $N_3 = 2^k = \text{poly}(1/\varepsilon) \leq \text{poly}(M)$, which is $\text{poly}(M)$.

We now turn to the decoding properties of this code. We start by showing that the concatenation of C_2 and C_3, denoted by C', is an approximate locally list-decodable (binary) code.

Lemma 4.1. $C' : \{0,1\}^{N_1} \to \{0,1\}^N$ is a $1/25$-approximate $(1/2-\varepsilon, \ell)$-locally-list-decodable code, where $\ell = \text{poly}(1/\varepsilon)$. The local decoder for C' can be implemented by constant-depth circuits of size $\text{poly}(\log M, 1/\varepsilon)$, with majority gates of fan-in $O(1/\varepsilon)$ (and AND/OR gates of unbounded fan-in).

Proof. By Theorem 4.10 (and the simplification assumption from Remark 2.2), there is a locally-list-decoder D_2 taking advice of size $\log(\ell_2)$, where $\ell_2 = \text{poly}(1/\varepsilon)$, such that for every $y \in \Gamma^{N_2}$ and for every $m \in \{0,1\}^{N_1}$ for which $\Delta_\Gamma(C_2(m), y) \leq 1 - \varepsilon^3/2$,

$$\exists a \in [\ell_2] \text{ s.t. } \Pr_{i \in_R [N_1]}[\Pr[D_2^y(a,i) = m[i]] > 9/10] \geq 24/25$$

Furthermore, D_2 can be implemented by a constant depth circuit of size $\text{poly}(\log M, 1/\varepsilon)$.

By Theorem 4.15, there is a locally-list-decoder D_3 taking advice of size $\log(\ell_3)$, where $\ell_3 = O(1/\varepsilon^2)$, such that for every $y \in \{0,1\}^{N_3}$ and for every $m \in \{0,1\}^k$ for which $\Delta(C_3(m), y) \leq 1/2 - \varepsilon/2$,

$$\exists a \in [\ell_3] \text{ s.t. } \forall i \in [k] \Pr[D_3^y(a,i) = m[i]] > 9/10$$

Furthermore, D_3 can be implemented by a constant depth circuit of size $\text{poly}(\log M, 1/\varepsilon)$, that uses majority gates of fan-in $O(1/\varepsilon)$.

The fact that the local-list-decoders for the two codes can be combined to obtain a local-list-decoder for the concatenated code (with list size that is the product of the two list sizes) is quite a standard argument. We refer the reader to [STV01] for the formal details. Here we just sketch the argument.

The decoder D' for the concatenated code C' roughly works as follows: it takes advice $(a_2, a_3) \in [\ell_2] \times [\ell_3]$. Given an index i, D' runs $D_2(a_2, i)$. Whenever the latter needs a (k-bit) symbol from its received word, D' runs $D_3(a_3, \cdot)$ to retrieve the whole symbol.

To analyze the correctness we argue as follows. For a received word $y \in \{0,1\}^N$ and a message $x \in \{0,1\}^{N_1}$ for which $\Delta(C'(x), y) \leq 1/2 - \varepsilon$, there are at least $\varepsilon/2$ symbols of $C_2(x)$ for which their C_3 encoding has $1/2 + \varepsilon/2$ agreement with the corresponding bits in y. Each one of these gives rise to a list of ℓ_3 possible symbols one of which is the correct one. By an averaging argument, there is a $a_3 \in [\ell_3]$, for which at least $\varepsilon/2 \cdot \varepsilon^2 = \varepsilon^3/2$ fraction of the symbols of $C_2(x)$ are such that the a_3'th element in the list produced by D_3 (with advice a_3) agrees with the corresponding symbol of $C_2(x)$. Since D_2 (with an appropriate advice a_2) can $1/25$-approximately recover from agreement $\varepsilon^3/2$, we get that the combined decoder with advice (a_2, a_3) recovers a string that has agreement $24/25$ with x.

The size of the decoders D_2, D_3 is $\text{poly}(\log M, 1/\varepsilon)$. Both are of constant depth where the latter uses majority gates of fan-in $\Theta(1/\varepsilon)$. Combining the two we get a constant-depth $(1/2 - \varepsilon, \text{poly}(1/\varepsilon))$-locally-list-decoder for the concatenated code of size $\text{poly}(\log M, 1/\varepsilon)$ with majority gates of fan-in $\Theta(1/\varepsilon)$. ∎

We now can describe the decoder D for C. On a received word $y \in \{0,1\}^N$, we run the local-decoder D_1 for C_1. Whenever it requires a bit from its received word (in $\{0,1\}^{N_1}$), we run the approximate local-decoder D' for C', with some advice string in $[\ell]$ (where $\ell = \ell_2 \cdot \ell_3$), to obtain a candidate for that symbol. If the received word has $1/2 - \varepsilon$ agreement with $C(x)$ (for some $x \in \{0,1\}^M$), then there exist an advice string $a \in [\ell]$ such that $D_2(a, \cdot)$ decodes correctly at least $24/25$ fraction of the symbols of $C_1(x)$. Thus, when D_1 receives symbols from $D_2(a, \cdot)$, it gets access to a word that has $24/25$ agreement with $C_1(x)$ and hence it correctly decodes every symbol in x.

Since the sizes of the two decoders is $\mathrm{poly}(\log M, 1/\varepsilon)$, and their depth is constant, then so are the size and depth of the combined decoder, and it uses majority gates of fan-in $\Theta(1/\varepsilon)$ because so does the decoder D_2. ■

As mentioned in Remark 4.9, we can obtain codes with quasi-polynomial rate that work for $\varepsilon = 1/M^\delta$. These are obtained by replacing the code C'_M in the proof of Theorem 4.10, which is a de-randomized direct-product code by [IJKW08], with their (not de-randomized) direct-product code. We state the parameters of these codes without a proof.

Theorem 4.16. *For every $1/M^\delta \leq \varepsilon = \varepsilon(M) < 1/2$ (where $\delta > 0$ is a constant), there exist a $(1/2 - \varepsilon, \mathrm{poly}(1/\varepsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \{0,1\}^{M^{O(\log(1/\varepsilon))}}\}_{M \in \mathbb{N}}$ with a local-decoder that can be implemented by a family of constant depth circuits of size $\mathrm{poly}(logM, 1/\varepsilon)$ that use majority gates of fan-in $\Theta(1/\varepsilon)$ (and AND gates of unbounded fan-in).*

5 Local-List-Decoding Requires Computing Majority

In this section we prove Theorem 1.7, showing that local list-decoding of binary error correcting codes from relative distance $1/2-\varepsilon$ essentially requires computing the majority function on $\Theta(1/\varepsilon)$ bits. In particular, this means that there are no constant-depth decoders for polynomial ε. This negative result is essentially tight (up to polynomial factors) given the construction of Theorem 1.5. We begin with a formal statement of the theorem:

Theorem 5.1 (Formal statement of Theorem 1.7)**.** *Let $\{C_M : \{0,1\}^M \to \{0,1\}^{N(M)}\}_{M\in\mathbb{N}}$ be a $(1/2-\varepsilon(M), \ell(M))$-locally-list-decodable code, such that $\ell(M) \leq 2^{\kappa\cdot M}$, and $1/N^{\delta_1} \leq \varepsilon(M) \leq \delta_2$ for universal constants $\kappa, \delta_1, \delta_2$. Let D be the local decoding machine, of size $S(M)$ and depth $d(M)$.*

Then, for every $M \in \mathbb{N}$, there exists a circuit A_M of size $\mathrm{poly}(S(M), \ell(M))$ and depth $O(d(M))$, that computes majority on $\Theta(1/\varepsilon(M))$ bits. The types of gates used by the circuit A_M are identical to those used by D. E.g., if D is an $AC^0[q]$ circuit, then so is A_M.

Proof Intuition for Theorem 1.7. Fix a message length M and $\varepsilon = \varepsilon(M)$. We will describe a circuit B with the stated parameters that decides the promise problem Π on inputs of length roughly $1/\varepsilon$. By Lemma 2.7 this will also give a circuit for computing majority.

We start with a simple case: assume that the (local) decoder D makes only non-adaptive queries to the received word. In this case we proceed using ideas from the proof of Theorem 6.4 in [Vio06]. Take m to be a message that cannot be even approximately decoded[11] from random noise with error rate $1/2$. Such a word exists by a counting argument. Let $C(m)$ be the encoding of m. Let $x \in \Pi_{Yes} \cup \Pi_{No}$ be a Π-instance of size $1/2\varepsilon$ (we assume w.l.o.g. throughout that $1/\varepsilon$ is an integer). B uses x to generate a noisy version of $C(m)$, by XORing each one of its bits with some bit of x that is chosen at random. It then uses D to decode this noisy version of $C(m)$. If $x \in \Pi_{No}$, this adds random noise (error rate $1/2$), and the decoding algorithm cannot recover most of m's bits. If $x \in \Pi_{Yes}$, then each bit is noisy with probability less than $1/2 - 2\varepsilon$, which means that w.h.p. the fraction of errors is at most $1/2-\varepsilon$, and the decoding algorithm successfully recovers every bit of m.

By comparing the answers of the decoding algorithm (or more precisely, every decoding algorithm in the list, by trying every possible advice) and the real bits of m in a small number of random locations, the algorithm B distinguishes w.h.p. whether $x \in \Pi_{Yes}$ or $x \in \Pi_{No}$.

Note, however, that B as described above is *not* a standard algorithm for Π. This is because we gave B access to the message m as well as its encoding. Both of these are strings that are much larger than we want B itself to be. So our next goal is to remove (or at least minimize) B's access to m and $C(m)$, making B a standard circuit for Π. Observe that B as described above distinguishes whether x is in Π_{Yes} or in Π_{No} with high probability over the choices of D's random coins, the random locations in which we compare D's answers against m, and the random noise generated by sampling bits from x. In particular, there exists a fixing of D's random string as well as the (small number of) testing locations of m that maintains the advantage in distinguishing whether x comes from Π_{Yes} or Π_{No}, where now the probability is only over the randomness used to sample bits from x. So now we can hardwire the bits of m used to test whether D decodes the noisy version of $C(m)$ correctly (i.e. we got rid of the need to store the whole string m). Furthermore, after we fix D's randomness, *by the fact that it is non-adaptive*, we get that the positions in which B queries the

[11] By this we mean that no decoder can recover (w.h.p.) a string that is, say, 1/3-close to m.

noisy $C(m)$ are now also fixed, and *independent of* x. So we also hardwire the values of $C(m)$ in these positions (and only these positions) into B. For any x, we now have all the information to run B and conclude whether x is in Π_{Yes} or Π_{No}.

Next we want to deal with adaptive decoders. If we proceed with the ideas described above, we run into the following problem: suppose the circuit has two (or more) levels of adaptivity. The queries in the second level do not only depend on the randomness of the decoder, but also on the values read from the received word at the first level, and in particular they also depend on *the noise*. The noise in our implementation depends on the specific Π-instance x. This means that we cannot hardwire the values of $C(m)$ that are queried at the second level because they depend on x!

To solve this problem, we analyze the behavior of the decoder when its error rate changes in the middle of its execution. Specifically, suppose that the decoder D queries the received word in d levels of adaptivity. For every $0 \leq k \leq d$, we consider the behavior of the decoder when up to level k we give it access to the encoded message corrupted with error-rate $1/2 - 2\varepsilon$, and above the k'th level we give it access to the encoded message corrupted with error-rate $1/2$. By a hybrid argument, there exists some level k, in which the decoder has a significant advantage in decoding correctly when up to the k'th level it sees error rate $1/2 - 2\varepsilon$ (and error-rate $1/2$ above it), over the case that up to the $(k-1)$'th level the error-rate is $1/2 - 2\varepsilon$ (and $1/2$ from k and up). We now fix and hardwire randomness for the decoder, as well as noise for the first $k-1$ levels (chosen according to error-rate $1/2 - 2\varepsilon$), such that this advantage is preserved. Once the randomness of D and the noise for the first $k-1$ levels are fixed, the queries at the k-th level (but not their answers) are also fixed. For this k-th level we can proceed as in the non-adaptive case (i.e. choose noise according to x and hardwire the fixed positions in $C(m)$). We now have to deal with queries above the k'th level. At first glance it is not clear that we have gained anything, because we still have to provide answers for these queries, and as argued above, these may now depend on the input x and therefore the query locations as well as the restriction of $C(m)$ to these locations cannot be hard-wired. The key point is that for these "top" layers the error rate has changed to $1/2$. So while we have no control on the query locations (as they depend on x) we do know their answers: they are completely random bits that have nothing to do with m or $C(m)$! Thus, B can continue to run the decoder, answering its queries (in the levels above the k'th) with random values. We thus obtain a circuit that decides membership in Π correctly with a small advantage. Since the number of adaptivity levels is only d (the circuit depth of the decoder), the distinguishing advantage of the k-th hybrid is at least $O(1/d)$, and in particular this advantage can now be amplified by using only additional depth of $O(\log(d))$. This gives a circuit that computes Π and concludes the proof. Due to space constraints we defer the formal proof of Theorem 1.7 to the full version of this paper [GR08].

Proof of Theorem 1.7. Fix $M \in \mathbb{N}$, $C = C_M : \{0,1\}^M \to \{0,1\}^{N(M)}$, $\varepsilon = \varepsilon(M)$, $\ell = \ell(M)$, $S = S(M)$ and $d = d(M)$ as in the statement of the theorem. We show how to use the decoder D to construct a circuit for computing Π on instances of size $1/\varepsilon$ (and thus also for computing majority, by Claim 2.7) as promised in the theorem statement.

Let us start with some notation. For an advice string $a \in [\ell]$, an index $i \in [M]$, and a received word $y \in \{0,1\}^N$, we denote by $D^y(a,i,r)$ an execution of the decoder D with advice a, randomness r, and (oracle) access to $y \in \{0,1\}^N$ to retrieve the i-th message bit (recall that we are working under the simplifying assumption from Remark 2.2). For $m \in \{0,1\}^M$ and $0 \leq \alpha \leq 1$, we use $\Gamma_\alpha(a,y,m)$ to denote the fraction of indices i in m that $D^y(a,i,r)$ recovers with probability at least

α (the probability is over D's randomness r). Formally:

$$\Gamma_\alpha(a, y, m) \stackrel{def}{=} \frac{1}{M}|\{i \in [M] : \Pr_r[D^y(a, i, r) = m[i]] \geq \alpha\}|$$

Let E_0 be the uniform distribution on $\{0,1\}^N$, and E_1 be the distribution over $\{0,1\}^N$ in which every bit is chosen (independently) to be 1 with probability $1/2 - 2\varepsilon$ and 0 otherwise.

First we show that there exists a message $m \in \{0,1\}^M$, such that if $C(m)$ is corrupted with completely random noise, then with probability 9/10 over the noise, for every advice string a, the decoder D cannot recover more than 3/5 of m's indices with probability greater than 3/5 (over its random coins).

Claim 5.2. *There exists a message $m \in \{0,1\}^M$ such that,*

$$\Pr_{e \leftarrow E_0}[\exists a \in [\ell] \text{ s.t. } \Gamma_{3/5}(a, C(m) \oplus e, m) > 3/5] \leq 1/10$$

Where the $\oplus$ operation between bit strings means bit-wise XOR.

Proof. The intuition is that if e is drawn from E_0 (error rate 1/2), then $C(m) \oplus e$ is independent of $C(m)$, and thus for most m's, all of the ℓ possible outputs of the decoder are far from m. Formally:

$$\begin{aligned}
&\Pr_{m \in \{0,1\}^M, e \leftarrow E_0} \quad [\exists a \in [\ell] \text{ s.t. } \Gamma_{3/5}(a, C(m) \oplus e, m) > 3/5] = \\
&\Pr_{m \in \{0,1\}^M, e \leftarrow E_0} \quad [\exists a \in [\ell] \text{ s.t. } \Gamma_{3/5}(a, e, m) > 3/5] = \\
&\Pr_{m \in \{0,1\}^M, e \leftarrow E_0} \quad [\exists a \in [\ell] \text{ s.t. } \frac{1}{M}|\{i \in [M] : \Pr_r[D^e(a, i, r) = m[i]] \geq 3/5\}| \geq 3/5]
\end{aligned}$$

Examining this last quantity, for any fixed error vector e and advice a, let m_a^e be the (single) message obtained by taking $m_a^e[i]$ to be the more probable answer (over r) of $D^e(a, i, r)$. Now, fixing e, and taking a random m, the probability that

$$\exists a \in [\ell] \text{ s.t. } \frac{1}{M}|\{i \in [M] : \Pr_r[D^e(a, i, r) = m[i]] \geq 3/5\}| \geq 3/5$$

is at most the probability that for the random m, for some $a \in [\ell]$, the fractional distance between m_a^e and m is at most 2/5. Denote by $Vol_{2/5}(M)$ the volume of the M-dimensional sphere of radius $2M/5$ (in the M-dimensional Hamming cube). Taking H to be the binary entropy function, the probability that there exists $a \in \ell$ such that m is 2/5-close to m_a^e is (by a union bound) at most:

$$\frac{\ell \cdot Vol_{2/5}(M)}{2^M} \leq \frac{\ell \cdot 2^{(H(2/5)+o(1))\cdot M}}{2^M} \leq \frac{1}{2^{\Omega(M)}} \leq 1/10$$

Where in the last inequality we assume $\ell \leq 2^{\kappa \cdot M}$ for a universal constant κ. We conclude that indeed:

$$\Pr_{m \in \{0,1\}^M, e \leftarrow E_0}[\exists a \in [\ell] \text{ s.t. } \Gamma_{3/5}(a, C(m) \oplus e, m) > 3/5] \leq 1/10$$

and thus certainly there *exists* an $m \in \{0,1\}^M$ for which

$$\Pr_{e \leftarrow E_0}[\exists a \in [\ell] \text{ s.t. } \Gamma_{3/5}(a, C(m) \oplus e, m) > 3/5] \leq 1/10$$

■

In contrast to the above claim, the decoding algorithm has the guarantee that for every message $m \in \{0,1\}^M$, with high probability over noise e of rate $1/2-2\varepsilon$ or less, there exists an advice string $a \in [\ell]$ such that when D is given this advice string and oracle access to the codeword $C(m)$ corrupted by e, it recovers every bit of m with probability 9/10.

Claim 5.3. *For every message $m \in \{0,1\}^M$:*

$$\Pr_{e \leftarrow E_1}[\exists a \in [\ell] \ s.t. \ \Gamma_{9/10}(a, C(m) \oplus e, m) = 1] > 9/10$$

Proof. Recall that the decoder D has the guarantee that if a codeword is corrupted in less than a $1/2-\varepsilon$-fraction of its coordinates, then for some $a \in [\ell]$, when D uses advice a it can recover each of the original message's coordinates with probability at least 9/10 (over its coins). It remain only to show that the probability that e drawn from E_1 corrupts more than a $1/2-\varepsilon$-fraction of $C(m)$'s coordinates is at most 1/10. This follows by a Chernoff bound, since e that is drawn from E_1 corrupts independently every coordinate of $C(m)$ with probability $1/2-1/2\varepsilon$. Then the probability that the fraction of coordinates corrupted is more than $1/2-1/\varepsilon$ is exponentially small in $1/\varepsilon$ (here we use that fact that $1/\varepsilon$ is significantly smaller than N, because $\varepsilon \geq 1/N^{\delta_1}$ for some universal constant $\delta_1 > 0$). In particular, for ε smaller than some universal constant $\delta_2 > 0$, this probability is indeed smaller than 1/10 as required. ∎

Fix m as in Claim 5.2. We define a probabilistic circuit A_1 that for $b \in \{0,1\}$ gets oracle access to a string $y = C(m) \oplus e$ where e is sampled from the distribution E_b. The goal of the circuit is to guess the value of b. We begin by constructing such a circuit that also gets oracle access to the string m. The algorithm is described in Figure 1.

Oracle access to: m and $y = C(m) \oplus e$ where $e \leftarrow E_b$.
Output: b.
The algorithm:
Let $q = \Theta(\log(\ell))$. For every $a \in [\ell]$ do the following in parallel:

1. Choose random indices $i_1^a, \ldots, i_q^a \in [M]$.
2. Choose random strings $r_1^a, \ldots, r_q^a$ for D.
3. For every $j \in [q]$ run $D^y(a, i_j^a, r_j^a)$ to obtain a prediction for the bit $m[i_j^a]$. If for at least $\frac{43}{50}$ of the j's, the prediction is equal to $m[i_j^a]$, output 1 and halt.

Otherwise (no $a \in [\ell]$ resulted in output 1), output 0 and halt.

Figure 1: Algorithm A_1

The algorithm A_1 (as described in Figure 1) can be implemented by a probabilistic oracle circuit of size $\text{poly}(S, \ell)$ and depth $O(d)$, where the circuit has oracle access to the message m and noisy codeword $C(m) \oplus e$. Denote by $\bar{r}$ the randomness used by A_1.

Claim 5.4.

$$\Pr_{e \leftarrow E_1, \bar{r}}[A_1^{m, C(m) \oplus e}(\bar{r}) = 1] - \Pr_{e \leftarrow E_0, \bar{r}}[A_1^{m, C(m) \oplus e}(\bar{r}) = 1] \geq 1/2$$

Proof. By Claim 5.3, when e is drawn from E_1, with probability 9/10, there exists $a \in [\ell]$ for which D (with advice a) successfully recovers each of m's indices with probability 9/10 (over its random

coins). In this case, when A_1 tries this a, with probability at least $1 - 1/\text{poly}(\ell)$, in at least $\frac{43}{50}$ of its q experiments it will successfully retrieve the proper bit of m (by a Chernoff bound). Taking a Union bound, we conclude that, when e is drawn from E_1, the probability that A_1 outputs 1 is at least 8/10.

By Claim 5.2, when e is drawn from E_0, with probability 9/10, for *every* $a \in [\ell]$, there exist a 2/5 fraction of m's indices, such that D (with advice a) fails to recover each one of them with probability at least 2/5 (over its coins). In this case, for any a in the execution of A_1, the probability of successfully recovering bits of m in a $\frac{43}{50}$ fraction of the experiments is at most $1/\text{poly}(\ell)$ (because at best, the decoder can recover with high probability 3/5 of the bits of m, and is expected, over its randomness to recover each of the remaining 2/5 bits with probability less than 3/5). Taking a Union bound, when e is drawn from E_0, the probability that A_1 outputs 1 is at most 2/10.

In conclusion:

$$\Pr_{e \leftarrow E_1, \bar{r}}[A_1^{m, C(m) \oplus e}(\bar{r}) = 1] - \Pr_{e \leftarrow E_0, \bar{r}}[A_1^{m, C(m) \oplus e}(\bar{r}) = 1] \geq 8/10 - 2/10 = 6/10 > 1/2$$

■

We now remove the need for oracle access to the message m. This can be done by fixing (for each $a \in [\ell]$) all of the $i_1^a, \ldots, i_q^a$ in the description of A_1, such that the difference in the probabilities of A_1 outputting 1 in Claim 5.4 is preserved (by averaging such a fixing exists). The values $m[i_1^a], \ldots, m[i_q^a]$ (for every $a \in [\ell]$) can then be hard-wired into the circuit A_1 (there are only $\text{poly}(\ell)$ of them). Let us call the new circuit A_2 which now has only oracle access to $C(m)$. We have,

$$\Pr_{e \leftarrow E_1, \bar{r}}[A_2^{C(m) \oplus e}(\bar{r}) = 1] - \Pr_{e \leftarrow E_0, \bar{r}}[A_2^{C(m) \oplus e}(\bar{r}) = 1] > 1/2 \tag{3}$$

The next step is to remove the oracle access to $C(m) \oplus e_b$ (these oracle queries are made by D). This is not straightforward since (as noted in the proof intuition) the queries of an adaptive decoder to the noisy codeword may depend on the noise, and through it (in our construction) on the input x itself. Since we do not know the query locations, we cannot hardwire the proper values of $C(m)$ into the circuit. We use a hybrid argument to overcome this difficulty. This involves further notation.

Assume that the decoder D asks its queries in d levels of adaptivity (d is a bound on its depth, so it is certainly a bound on the number of adaptive levels). For d distributions, $G^1, \ldots, G^d$ on $\{0,1\}^N$, we denote by $A_2^{C(m) \oplus G^1, \ldots, G^d}(\bar{r})$ the output of A_2, with randomness $\bar{r}$, where queries to the noisy codeword are answered as follows: for every adaptivity level $k \in [d]$ of the decoder D, sample $e^k \leftarrow G^k$. If in its k-th level, D queries the codeword in position $j \in [N]$, then the answer is $C(m)[j] \oplus e^k[j]$.

Note that if we use an oracle as described above (that generates a different noise vector for each adaptivity level), then if the same query is asked in different levels, the answers may be inconsistent. We want all answers to be consistent between the adaptivity levels and across all of A_2's executions of D (note that consistency across executions is important because the list-decoding guarantee is against a single fixed noise vector). To guarantee consistency, we modify A_2 so that the answers to queries across different executions (and within each execution) are always consistent; if k is the *first* execution in the *minimal* level in which query j is made (across the parallel executions), then the answer to query j is always $C(m)[j] \oplus e^k[j]$. We note that this consistency guarantee can be

realized with an $\mathcal{AC}^0$ circuit, by always answering a query with the answer given to that query in the lexicographically first level and execution number.

Now, for every $0 \leq k \leq d$, we define

$$O^k \stackrel{def}{=} C(m) \oplus \overbrace{E_1, \ldots, E_1}^{k} \overbrace{E_0, \ldots, E_0}^{d-k}$$

Consider running A_2 with oracle O^k. That is, for the first k levels we give A_2 access to $C(m)$ corrupted with error rate $1/2 - 2\varepsilon$ and for the last $d-k$ levels we give it access to $C(m)$ corrupted with error rate $1/2$. By (3):

$$\Pr_{\bar{r},E_1,\ldots,E_1}[A_2^{O^d}(\bar{r}) = 1] - \Pr_{\bar{r},E_0,\ldots,E_0}[A_2^{O^0}(\bar{r}) = 1] \geq \frac{1}{2}$$

This inequality holds because for the oracles O^0 and O^d, all the error vectors (in the different levels) have the same error rate. In this case, A_2 with the above "consistency modification", behaves identically to A_1 (with the hard-wired bits of m) with that same error rate. It follows, by triangle inequality, that there exists $1 \leq k \leq d$, such that,

$$\Pr_{\bar{r},E_0,\ldots,E_0,E_1,\ldots,E_1}[A_2^{O^k}(\bar{r}) = 1] - \Pr_{\bar{r},E_0,\ldots,E_0,E_1,\ldots,E_1}[A_2^{O^{k-1}}(\bar{r}) = 1] \geq \frac{1}{2d} \tag{4}$$

Fix such a k. Consider the circuit A obtained from A_2 as follows: Fix $\bar{r}$, as well as the noise for the answers of the oracle on the first $k-1$ levels, such that the advantage in Inequality (4) is preserved. After doing this, all the queries as well as their answers for the first $k-1$ levels are fixed. Hardwire all of them into the circuit (these are $\text{poly}(S, \ell)$ bits). Also, the queries (but not their answers) in the k'th level are fixed. Hardwire these queries into the circuit, as well as the values of $C(m)$ in these positions.

We now use A to answer a new guessing game. It is given access to a sample $e \leftarrow E_b$ ($b \in \{0, 1\}$) and it has to guess the value of b. It does so by simulating A_2 with the fixed randomness $\bar{r}$, answering oracle queries as follows: for the first $k-1$ levels it uses the fixed queries and their answers. For level k, if A_2 queries the received word in position j (which is now fixed), A returns as an oracle answer the value $C(m)[j] \oplus e[j]$ (recall that $C(m)[j]$ is hardwired). For the levels above k, A returns random bits (uniformly and independently distributed) as oracle answers. Note that throughout A, just like A_2, guarantees consistency of answers to D's oracle queries across the parallel executions and adaptivity levels.

Since A_2 is an oracle circuit of size $poly(S, \ell)$ and depth $O(d)$, then so is A. Also, it is clear that A simulates $A_2^{O^k}$ when $b = 1$ and $A_2^{O^{k-1}}$ when $b = 0$ (with fixed values that maximize the gap in (4)). Let $\bar{r}'$ be the randomness of A. We have,

$$\Pr_{e \leftarrow E_1, \bar{r}'}[A^e(\bar{r}') = 1] - \Pr_{e \leftarrow E_0, \bar{r}'}[A^e(\bar{r}') = 1] \geq \frac{1}{2d} \tag{5}$$

Let

$$\gamma \stackrel{def}{=} \Pr_{e \leftarrow E_1, \bar{r}'}[A^e(\bar{r}') = 1] = \Pr[A_2^{O^k} = 1]$$

We are finally ready to describe a circuit B that computes Π correctly on instances of length $1/2\varepsilon$ with a small advantage (that will later be amplified). We assume w.l.o.g. that $1/2\varepsilon$ is an even

integer. On input $x \in \Pi_{yes} \cup \Pi_{No}$ of length $1/2\varepsilon$, B runs A while simulating the noise $e \leftarrow E_b$ as follows: whenever A queries e in position j, B chooses uniformly $i \in [1/2\varepsilon]$ and returns the bit $x[i]$. At the end of the execution, B returns the same answer as A does.

B is also a circuit of size $\text{poly}(S, \ell)$ and depth $O(d)$ (inherited from A). If $x \in \Pi_{Yes}$, then $\Pr_i[x[i] = 1] = 1/2 - 2\varepsilon$, and the simulated oracle is distributed identically to a sample from E_1. On the other hand, if $x \in \Pi_{No}$, then $\Pr_i[x[i] = 1] = 1/2$, and the simulated oracle is distributed identically to a sample from E_0. We conclude from Inequality (5):

Claim 5.5. *If $x \in \Pi_{Yes}$, $\Pr[B(x) = 1] \geq \gamma$. And if $x \in \Pi_{No}$, $\Pr[B(x) = 1] \leq \gamma - \frac{1}{2d}$.*

Finally, we amplify the success probability of B. This can be done by hard-wiring γ in the circuit, and running B a $\text{poly}(d)$ number of times (in parallel) with independent random coins. If at least $\gamma - \frac{4}{10d}$ of the executions return 1, then return 1, and otherwise 0. By Claim 5.5 and a Chernoff bound, this amplified version of B computes Π correctly on instances of size $1/2\varepsilon$ with probability of error at most $1/10$. Furthermore, it is a circuit of size $\text{poly}(S, \ell)$ and depth $O(d)$ (note that counting the number of 1-answers in the $\text{poly}(d)$ executions that are run for the final amplification step only requires additional depth $O(\log(d))$).

■

By using known lower bounds for computing the majority function by $AC^0[q]$ circuits (for a prime q) [Raz87, Smo87], we obtain the following corollary.

Corollary 5.6. *Let $\{C_M : \{0,1\}^M \to \{0,1\}^{N(M)}\}_{M \in \mathbb{N}}$ be a $(1/2 - \varepsilon, \ell)$-locally-list-decodable code (where ε is in the range specified in Theorem 1.7) with a decoder that can be implemented by a family of $AC^0[q]$ circuits of size $s = s(M)$ and depth $d = d(M)$. Then $s = 2^{(1/\varepsilon)^{\Omega(1/d)}}/\text{poly}(\ell)$.*

6 Hardness Amplification

What is Hardness Amplification? Functions that are hard to compute *on the average* (by a given class of algorithms or circuits) have many applications, for example in cryptography or for de-randomization via the construction of pseudo-random generators (the "hardness vs. randomness" paradigm [BM84, Yao82, NW94]). Typically, for these important applications, one needs a function that no algorithm (or circuit) in the class can compute it on random inputs much better than a random guess. Unfortunately, however, it is often the case that one does not have or cannot assume access to such a "hard on the average" function, but rather only to a function that is "somewhat hard": every algorithm in the class fails to compute it and errs, but only on relatively few inputs (e.g. a small constant fraction, or sometimes even just a single input from every input length). In order to bridge this "hardness gap", an approach that has been used (very successfully) is to find a way to convert "somewhat hard" functions to functions that are "very hard" (on the average). Procedures that attain this goal are called *hardness amplification* procedures or reductions.

Let us be more precise. We say that a Boolean function $f : \{0,1\}^* \to \{0,1\}$ is δ-hard on the average for a circuit class $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ (where circuits in the set $\mathcal{C}_n$ have input length n), if for every large enough n, for every circuit $C_n \in \mathcal{C}_n$;

$$\Pr_{x\in_R U_n}[C_n(x) = f(x)] \leq 1 - \delta$$

The task of obtaining from a function f that is δ-hard for a class $\mathcal{C}$, a function f' that is δ'-hard for the class $\mathcal{C}$, where $\delta' > \delta$ is called hardness amplification from δ-hardness to δ'-hardness (against the class $\mathcal{C}$). Typical values for δ are small constants (close to 0), and sometimes even 2^{-n}, in which case the hardness amplification is from worst-case hardness. Typical values for δ' (e.g. for cryptographic applications) are $1/2 - n^{-\omega(1)}$.

The most commonly used approach to prove hardness amplification results is via reductions, showing that if there is a sequence of circuits in $\mathcal{C}$ that computes f' on more than a $1-\delta'$ fraction of the inputs, then there is a sequence of circuits in $\mathcal{C}$ that computes f on more than $1-\delta$ fraction of the inputs. An important family of such reductions are so-called fully-black-box reductions which we define next.

Definition 6.1. A (δ, δ')-fully-black-box hardness amplification from input length k to input length $n = n(k, \delta, \delta')$, is defined by an oracle Turing machine Amp that computes a Boolean function on n bits, and an oracle Turing machine Dec that takes non-uniform advice of length $a = a(k, \delta, \delta')$. It holds that For every $f : \{0,1\}^k \to \{0,1\}$, for every $A : \{0,1\}^n \to \{0,1\}$ for which

$$\Pr_{x\in_R U_n}[A(x) = Amp^f(x)] > 1 - \delta'$$

there is an advice string $\alpha \in \{0,1\}^a$ such that

$$\Pr_{x\in_R U_k}[Dec^A(\alpha, x) = f(x)] > 1 - \delta$$

where $Dec^A(\alpha, x)$ denotes running Dec with oracle access to A on input x and advice α.

If Dec does not take non-uniform advice ($a = |\alpha| = 0$), then we say that the hardness amplification is *uniform*.

The Complexity of Hardness Amplification. We now elaborate on the role that the complexity of Dec plays in hardness amplification. Recall that hardness amplification is used to amplify the average-case hardness of functions that are somewhat hard. In particular, suppose we want to obtain from a function $f : \{0,1\}^k \to \{0,1\}$ that is δ-hard against some class (of algorithms or circuits) $\mathcal{C}$, a function $f' : \{0,1\}^n \to \{0,1\}$ that is δ'-hard against $\mathcal{C}$, using a hardness amplification procedure as defined in Definition 6.1. For this application, we need a (δ, δ')-fully-black-box hardness amplification from length k to length n (as above), such that Dec itself (as a machine with non-uniform advice) is in the class $\mathcal{C}$. To see this, set $f' = Amp^f$. Then by contradiction, if there is $A \in \mathcal{C}$ that computes f' on more than $1-\delta'$ fraction of the instances of length n, then $Dec^A(\alpha, \cdot)$ computes f on more than $1-\delta$ fraction of the instances of length k. Furthermore, $Dec^A(\alpha, \cdot) \in \mathcal{C}$ (here we assume that $\mathcal{C}$ is informally "closed under oracle access"), which is a contradiction to the δ-hardness of f. To summarize, the complexity of Dec determines against which class of algorithms or circuits the hardness amplification can be used. In particular, if one wants to use such hardness amplification to amplify hardness against uniform classes of algorithms or circuits, then the hardness amplification must be uniform.

We note that the question of finding functions that are average-case-hard for low complexity classes, such as $AC^0[q]$, is of central importance for de-randomizing these classes [NW94]. This motivates the study of hardness amplification against such classes, especially since these are the only classes for which (unconditional) mildly average hardness results are known [Raz87, Smo87], and thus there is clear hope of unconditional de-randomization. We now elaborate: a function f that is very hard on the average (at least $1/2 + 1/\mathrm{poly}n$) for a class can be used in the Nisan-Wigderson construction [NW94], to obtain efficient pseudo-random generators that fool statistical tests in the class. This, in turn, can give a de-randomization of the class. Unfortunately, for classes such as $AC^0[q]$, no such hardness results are known: [Raz87, Smo87] only give constant hardness (smaller than $1/2$) of the mod p function for a prime $p \neq q$. Consequently, we do not know how to unconditionally de-randomize probabilistic $AC^0[q]$ circuits, even using sub-exponential size deterministic $AC^0[q]$ circuits.

It is well known [STV01, TV07, Tre03, Vio03] that there is a tight connection between $(2^{-k}, \delta')$-fully-black-box hardness amplification (or in other words worst-case to average-case reductions) and binary locally (list) decodable codes. We state this fact without proof.

Proposition 6.1. There is a $(1/2 - \varepsilon, \ell)$-locally-list-decodable code $Enc : \{0,1\}^K \to \{0,1\}^N$ with a decoder D, if and only if there is a $(2^{-k}, 1/2 - \varepsilon)$-fully-black-box hardness amplification from length $k = \log K$ to length $n = \log N$ defined by Amp and Dec, that takes $a = \log \ell$ bits of advice, where Amp is Enc and Dec is D.

Our results, both positive and negative, on constructing locally list-decodable binary codes, lead to progress on the complexity of hardness amplification.

New worst-case to average-case reductions for low complexity classes. As stated above, it is well known that explicit binary locally list-decodable codes with polynomial rate (and even quasi-polynomial rate) imply worst-case to average-case reductions for the class $\mathcal{EXP}$ [BFNW93, IW97, STV01]. To see this, consider such a code $C : \{0,1\}^M \to \{0,1\}^N$ that can be locally and list decoded from agreement $1/2+\varepsilon$. Let $f : \{0,1\}^* \to \{0,1\}$ be an $\mathcal{EXP}$-complete function, and let T_f^m be its $M = 2^m$-bit truth-table for input length m. Now consider the function $\hat{f}$ whose truth table for input length $cm = \log N$ (for some constant c depending on the rate of the code) is $C(T_f^m)$.

$\hat{f} \in \mathcal{EXP}$ because the code is explicit and T_f^m can be computed in time 2^{m^d} (for some constant d). Now suppose that there is an efficient algorithm A that computes $\hat{f}$ correctly on a $1/2 + \varepsilon$ fraction of the inputs of length cm. Then the truth table of A at that length has $1/2+\varepsilon$ agreement with $C(T_f^m)$. This together with the decoder for C gives an algorithm B that solves f correctly on every input (of length m) given some non-uniform advice: indeed, by letting A answer the queries of the decoder, we can generate a list of circuits, one of which solves f correctly everywhere, and the advice bits give the index of this circuit in the list. Now, since both the decoder and A are efficient, we obtain an efficient algorithm that solves f correctly on the worst-case. Thus there is an equivalence between the worst-case and average-case hardness of $\mathcal{EXP}$-complete languages, against efficient algorithms.

In the argument above, B runs both the algorithm A (whose efficiency we determine by hypothesis) and the decoder (whose efficiency is a property of the code). Thus the efficiency of the decoder determines the efficiency of the worst-case to average-case reduction, and hence the class of efficient algorithms against which we can establish worst-case/average-case equivalence in $\mathcal{EXP}$. By using our construction from Theorem 1.5, we obtain a reduction in non-uniform $\mathcal{AC}^0$ for polylogarithmic ε. However, we can do better. By using ideas from Trevisan and Vadhan [TV07], we can obtain a reduction in *uniform* $\mathcal{AC}^0$. They show how to remove the non-uniform advice bits (assuming that there are not too many of them) by using the fact the $\mathcal{EXP}$-complete languages have *instance checkers*. Roughly speaking, an instance checker for a language L, is an efficient probabilistic oracle algorithm, that on input x computes $L(x)$ when given access to an oracle that computes L correctly, and when given a faulty oracle, either detects that the oracle does not compute L correctly or it still outputs $L(x)$. We refer the reader to [TV07] for the formal definition. Using the fact that there are very efficient instance checkers for $\mathcal{EXP}$-complete languages, we obtain the following theorem.

Theorem 6.2. *Let $\mathcal{C}$ be a class of algorithms (or Boolean circuits) that can compute probabilistic uniform $\mathcal{AC}^0$ circuits. Then for every $\mathcal{EXP}$ function $f : \{0,1\}^* \to \{0,1\}$, there is an $\mathcal{EXP}$ function $\hat{f} : \{0,1\}^* \to \{0,1\}$ such that: for every large enough m, if there is no algorithm (or family of circuits) in the class $\mathcal{C}$ that computes f at length m correctly in the worst-case, then there is no algorithm (or family of circuits) in the class $\mathcal{C}$ that can compute $\hat{f}$ at length $n = \mathrm{poly}(m)$ correctly on at least a $1/2 + 1/(\log m)^\alpha$ fraction of the inputs, where $\alpha > 0$ is a universal constant.*

Proof. Consider the language $L = \{(M, x, c) : M$ is a deterministic TM that accepts x within c steps, where c is in binary representation$\}$. It is $\mathcal{EXP}$-complete under $\mathcal{AC}^0$ reductions. So it is enough to show a worst-case to average-case reduction with the specified parameters from this language. We need the following easy claim.

Claim 6.3. *L has an instance checker (with exponentially small error) that can be implemented in $\mathcal{AC}^0$.*

Proof. One can apply our general approach presented in this paper to obtain the claim. However there is a much simpler way. It is well known that a complete language for a given class (of languages) has an instance checker, if it has a Probabilistic Checkable Proof in which the problem of computing every given bit in the proof can be done within the class. In particular $\mathcal{EXP}$-complete languages have this property. The computation of the instance checker involves running the PCP verifier as well as the reduction to the complete language.

Now take the PCP proof for an instance of L, and append to it for every possible randomness of the verifier, the tableau of the verifier's computation (given the input, the randomness and the

bits read from the original proof). This is still a PCP proof with the property that every bit of it can be computed in $\mathcal{EXP}$. But now the verifier simply needs to check the correctness of the tableau and this can be done in $\mathcal{AC}^0$. Finally since L is complete under $\mathcal{AC}^0$ reductions, we obtain an instance checker for L that can be implemented in $\mathcal{AC}^0$. ∎

We now proceed as in the outline above. Let T_m^L be the $M = 2^m$-bit truth-table of L for input length m. Consider the function $\hat{f}$ whose truth-table for input length n is $C(T_m^L)$, where C is the code from Theorem 1.5. We set ε to be $1/(\log m)^\alpha = 1/(\log\log M)^\alpha$, where $\alpha > 0$ is a constant that ensures that the $\mathcal{AC}^0$ decoder for C is of size $\operatorname{poly}\log M = \operatorname{poly}(m)$. By the parameters of C, $n = \log|C(T_m^L)| = O(m)$.

We prove the contrapositive. Suppose there is an algorithm A in the class $\mathcal{C}$ that computes $\hat{f}$ correctly on $1/2 + 1/(\log m)^\alpha$ fraction of the inputs of length n. Then we can run the decoder for C with oracle access to A and obtain $\operatorname{poly}\log m$ algorithms (that use A as an oracle) one of which computes L correctly (everywhere) on input length m. Since $A \in \mathcal{C}$ and the decoder is in $\mathcal{AC}^0$, we conclude that every algorithm in the list is in $\mathcal{C}$. Now to solve a given input of length m, we run the $\mathcal{AC}^0$ instance checker for L with each one of the algorithms in the list as an oracle (we do that in parallel and with independent random coins). Finally, we compute the approximate majority of the answers we receive from the instance checker. The whole procedure is in uniform $\mathcal{AC}^0$ making oracle calls to A, and hence it is in $\mathcal{C}$. ∎

Negative Results. Using this equivalence of fully black-box hardness amplification and local list-decoding, together with Theorem 1.7, we can show (informally) that worst-case to average-case hardness amplification with small non-uniform advice requires computing majority. This is stated formally in the theorem below:

Theorem 6.4. *If there is a $(2^{-k}, 1/2-\varepsilon(k))$-fully-black-box hardness amplification from length k to length $n(k)$ where Dec takes $a(k)$ bits of advice and can be implemented by a circuit of size $s(k)$ and depth $d(k)$, then for every $k \in \mathbb{N}$ there exists a circuit of size $\operatorname{poly}(s(k), 2^{a(k)})$ and depth $O(d(k))$, that computes majority on $O(1/\varepsilon(k))$ bits.*

It is known [Raz87, Smo87] that low complexity classes *cannot* compute majority. Thus, Theorem 6.4 shows limits on the amount of hardness amplification that can be achieved by fully-black-box worst-case to average-case reductions (that do not use too many bits of advice), in which *Dec* can be implemented in low-level complexity classes. I.e. classes that cannot compute majority (e.g. $\mathcal{AC}^0$ and $AC^0[q]$). The reason is that if there exists hardness amplification for which *Dec* is in such a class, then by Theorem 6.4 there must be a circuit family in the same class for majority, contradicting known circuit lower bounds [Raz87, Smo87]. In particular, the theorem implies that there are no uniform (or even $O(\log 1/\varepsilon)$-non-uniform) $(2^{-k}, 1/2-\varepsilon)$-fully-black-box worst-case to average-case reductions for ε smaller than $1/\operatorname{poly}\log k$, where *Dec* is a $AC^0[q]$ circuit (for a prime q) of size $\operatorname{poly}(k, 1/\varepsilon)$. This should be contrasted with [GGH+07] who showed such a fully-black-box reduction (with *Dec* in $\mathcal{AC}^0$) for $\varepsilon \geq 1/\log^\beta k$, where β is a universal constant.

Finally, we note that the worst-case lower bounds (which are actually mildly average-case lower bounds) of [Raz87, Smo87] hold against *non-uniform* $AC^0[q]$. This means that it may be possible to get the average-case hardness required for pseudo-randomness by using a lot of non-uniformity in a fully-black-box reduction (i.e. a reduction in which *Dec* takes $\operatorname{poly}(k)$ bits of advice). Shaltiel

and Viola [SV08] rule out such non-uniform fully-black-box reductions in the special case that Dec has only non-adaptive access to A.

Extensions. Theorem 6.4 can be extended in two ways: first to rule out hardness amplification from mildly hard functions (and not necessarily worst-case hard) to very hard functions, and second to rule out *not necessarily fully* black-box hardness amplification.

Let us start with the first direction. Proposition 6.1 can be extended to show a similar equivalence between δ-approximate locally $(1/2-\varepsilon,\ell)$-list-decodable codes to $(\delta, 1/2-\varepsilon)$-fully-black-box hardness amplification (with the same translations between the parameters). Let $0<\alpha<1/2$ be an arbitrary constant. Theorem 1.7 can be extended to show that a $1/2-\alpha$-approximate locally $(1/2-\varepsilon,\ell)$-list-decodable code implies circuits for majority with the same parameters as in the statement of Theorem 6.4.[12] Putting the two together we obtain the following.

Theorem 6.5. *Let $0<\alpha<1/2$ be an arbitrary constant. If there is a $(1/2-\alpha, 1/2-\varepsilon(k))$-fully-black-box hardness amplification from length k to length $n(k)$, where Dec takes $a(k)$ bits of advice and can be implemented by a circuit of size $s(k)$ and depth $d(k)$, then for every $k\in\mathbb{N}$ there exist a circuit of size $\mathrm{poly}(s(k), 2^{a(k)})$ and depth $O(d(k))$, that computes majority on $1/\varepsilon(k)$ bits.*

We conclude with an informal discussion about *not necessarily fully* black-box hardness amplification. Note that in definition 6.1, the hardness amplification is required to work for every function f. A more relaxed notion is (not necessarily fully) black-box reductions:

Definition 6.6. A (δ,δ')-black-box hardness amplification from $f:\{0,1\}^k\to\{0,1\}$ to $f':\{0,1\}^n\to\{0,1\}$ is defined by an oracle Turing machine Dec that takes non-uniform advice of length $a=a(k,\delta,\delta')$ and the following holds; for every $A:\{0,1\}^n\to\{0,1\}$ for which

$$\Pr_{x\in_R U_n}[A(x)=f'(x)]>1-\delta'$$

there is an advice string $\alpha\in\{0,1\}^a$ such that

$$\Pr_{x\in_R U_k}[Dec^A(\alpha,x)=f(x)]>1-\delta$$

This is relaxation of fully-black-box hardness amplification. In this case, the hardness amplification is not required to work for *any* function, but only for a *specific and known* function. Suppose we have a function f that we already know is worst-case hard, or even δ-hard on the average, against a low level class such as $AC^0[q]$. Perhaps we can use specific properties of the function f (e.g. random self-reducability) to construct a function f', such that there is a $(\delta, 1/2-1/poly(n))$-black-box hardness amplification from f to f' that *can* be implemented by $AC^0[q]$ circuits. This would not be a fully-black-box hardness amplification result, but it certainly suffices for de-randomization applications (in fact, usually for de-randomization one uses a specific and explicit hard function).

We note that the results of Theorems 6.4 and 6.5 can be extended to show that if a function f is δ-hard on the average for a low complexity class, and furthermore, there is a uniform (or even somewhat non-uniform) $(\delta+1/\mathrm{poly}\log(k), 1/2-\varepsilon(k))$-hardness amplification from f to *any* other

[12] The proof follows the outline of the proof of Theorem 1.7 using the fact that from error rate $1/2$ we cannot recover more than $1/2+\alpha/2$ of the bits of the message m, while from error rate $1/2-\varepsilon$ we can recover at least $1/2+\alpha$ of the bits. So by sampling bits from m we can distinguish between the two cases.

function f', where Dec is of size $s(k)$ and depth $d(k)$, then there exists a circuit of similar size and depth that computes majority on $O(\varepsilon(k))$-bit inputs.

The basic idea is similar to the proof of Theorem 6.4. The decoder cannot, given an oracle for f' that is only correct with probably $1/2$ (over the inputs), recover f with probability greater than $1-\delta$. This is because doing so would contradict the hardness of f: computing any f' with error rate $1/2$ is computationally easy, so the oracle can be simulated by an $\mathcal{AC}^0$ circuit, and we get a circuit for computing f. On the other hand, the reduction *does* recover from error rate $1/2-\varepsilon(k)$, computing f correctly with probability $1-\delta-\operatorname{poly}\log(k)$. This gives a distinguisher between error rates $1/2$ and $1/2-\varepsilon(k)$, which in turn (as in the proof of Theorem 6.4) leads to an algorithm for computing majority on $O(\varepsilon(k))$ bits. The full details are omitted.

7 Acknowledgements

We thank Salil Vadhan for his assistance, encouragement and insightful comments. Thanks to Russell Impagliazzo for helpful discussions and for supplying us with an early manuscript of [IJKW08]. We also thank Ronen Shaltiel and Emanuele Viola for helpful discussions on the topics addressed in this work, and for supplying us with an early manuscript of [SV08]. Finally, we thank anonymous referees for their many helpful comments.

References

[AB84] Miklós Ajtai and Michael Ben-Or. A theorem on probabilistic constant depth computation. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 471–474, 1984.

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–, 1992.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[Ajt93] Miklós Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances in computational complexity theory*, pages 1–20. Amererican Mathematical Society, 1993.

[AS00] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, Inc., 2000.

[Bab87] László Babai. Random oracles separate pspace from the polynomial-time hierarchy. *Inf. Process. Lett.*, 26(1):51–53, 1987.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. Bpp has subexponential time simulations unless exptime has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[Bli86] V. M. Blinkovsky. Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission*, 22(1):7–19, 1986.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[DJK+02] Amit Deshpande, Rahul Jain, Telikepalli Kavitha, Jaikumar Radhakrishnan, and Satyanarayana V. Lokam. Better lower bounds for locally decodable codes. In *IEEE Conference on Computational Complexity*, pages 184–193, 2002.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, 1981.

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007.

[GGH+08] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. A (de)constructive approach to program checking. In *STOC*, pages 143–152, 2008.

[GKST06] Oded Goldreich, Howard J. Karloff, Leonard J. Schulman, and Luca Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. *Computational Complexity*, 15(3):263–296, 2006.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GR08] Dan Gutfreund and Guy N. Rothblum. The complexity of local list decoding. In *APPROX-RANDOM*, pages 455–468, 2008.

[GV04] Dan Gutfreund and Emanuele Viola. Fooling parity tests with parity gates. In *APPROX-RANDOM*, pages 381–392, 2004.

[GV05] Venkatesan Guruswami and Salil P. Vadhan. A lower bound on list size for list decoding. In *APPROX-RANDOM*, pages 318–329, 2005.

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *In Proceedings of STACS 2006*, pages 672 – 683, 2006.

[IJK06] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Approximately list-decoding direct product codes and uniform hardness amplification. In *FOCS*, pages 187–196, 2006.

[IJKW08] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct product theorems: simplified, optimized, and derandomized. In *STOC*, pages 579–588, 2008.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[IW97] Russell Impagliazzo and Avi Wigderson. $= BPP$ if requires exponential circuits: Derandomizing the xor lemma. In *STOC*, pages 220–229, 1997.

[KdW04] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. Comput. Syst. Sci.*, 69(3):395–420, 2004.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC*, pages 80–86, 2000.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[Oba02] Kenji Obata. Optimal lower bounds for 2-query locally decodable linear codes. In *RANDOM*, pages 39–50, 2002.

[Raz87] Alexander A. Razborov. Lower bounds on the dimension of schemes of bounded depth in a complete basis containing the logical addition function. *Akademiya Nauk SSSR. Matematicheskie Zametki*, 41(4):598–607, 623, 1987.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the xor lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

[SV08] Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. In *STOC*, pages 589–598, 2008.

[Tre03] Luca Trevisan. List-decoding using the xor lemma. In *FOCS*, pages 126–135, 2003.

[Tre04] Luca Trevisan. Some applications of coding theory in computational complexity. *CoRR*, cs.CC/0409044, 2004.

[TV07] Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007.

[Vio03] Emanuele Viola. Hardness vs. randomness within alternating time. In *IEEE Conference on Computational Complexity*, pages 53–, 2003.

[Vio05] Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *Computational Complexity*, 13(3-4):147–188, 2005.

[Vio06] Emanuele Viola. *The complexity of hardness amplification and derandomization.* PhD thesis, Harvard University, 2006.

[WdW05] Stephanie Wehner and Ronald de Wolf. Improved lower bounds for locally decodable codes and private information retrieval. In *ICALP*, pages 1424–1436, 2005.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1), 2008.