The Complexity of Local List Decoding

Dan Gutfreund [*]
Math. Dept. and CSAIL, MIT,
danny@math.mit.edu

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

Abstract

We study the complexity of locally list-decoding binary error correcting codes with good parameters (that are polynomially related to information theoretic bounds). We show that computing majority over $\Theta(1/\epsilon)$ bits is essentially equivalent to locally list-decoding binary codes from relative distance $1/2 - \epsilon$ with list size at most poly$(1/\epsilon)$. That is, 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 $\Theta(1/\epsilon)$ bits. On the other hand, there is an explicit locally list-decodable code with these parameters that has a very efficient (in terms of circuit size and depth) local-decoder that uses majority gates of fan-in $\Theta(1/\epsilon)$.

Using known lower bounds for computing majority by constant depth circuits, our results imply that every constant-depth decoder for such a code must have size almost exponential in $1/\epsilon$ (this extends even to sub-exponential list sizes). This shows that the list-decoding radius of the constant-depth local-list-decoders of Goldwasser *et al.* [STOC07] is essentially optimal.

Using the tight connection between locally-list-decodable codes and hardness amplification, we obtain similar limitations on the complexity of uniform (and even somewhat non-uniform) fully-black-box worst-case to average-case reductions. Very recently, Shaltiel and Viola [SV08] independently obtained similar limitations for *completely* non-uniform fully-black-box worst-case to average-case reductions, but only for the special case that the reduction is *non-adaptive*. Our results apply also to *adaptive* reductions.

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 Σ (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

[*]Research was partially supported by NSF grant CCF-0514167. Part of this research was done while the author was at Harvard University and supported by ONR grant N00014-04-1-0478 and NSF grant CNS-0430336.

[†]Research supported by NSF grants CCF-0729011, CFF-0635297, and CNS-0430336.

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.

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 [STV99] (we elaborate on this application 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.

1.1 This Work

In this work we study the complexity of locally list decoding binary codes (i.e. where the alphabet is $\{0,1\}$). In particular, we characterize the computational complexity of such decoders, for codes that have "good" parameters. Here by "good" parameters we mean parameters that are polynomially related to the information theoretic bounds (we elaborate on this choice below).[1]

We proceed more formally. Let $C : \{0,1\}^M \rightarrow \{0,1\}^N$ 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. Each D_a takes as input an index $j \in [M]$ and tries to output the j-th bit of the message. The deocder is a $(1/2-\epsilon, \ell)$-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 $1/2-\epsilon$, with high probability at least one of the D_a's successfully decodes *every* bit of the message y. Note that here $1/2-\epsilon$ refers to the "noise rate" (or the list-decoding radius) from which the decoder recovers, and ℓ is the "list size": the number of decoding circuits, one of which makes the decoder recover every index correctly (with high probability).

In this work 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 bit. 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 Definition 1 for a formal definition of locally list decodable codes. We would like to point out that we use $(1/2-\epsilon, \ell)$ to denote the relative distance and list size, whereas previous work (e.g. [STV01]) used (ϵ, ℓ) to denote the same thing (for binary codes). We find this notation more useful, especially when we work with non-binary codes (which come up in our construction).

[1] 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].

[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 1.

It is well known that for every $(1/2-\epsilon, \ell)$-locally-list-decodable code, it must hold that $\ell = \Omega(1/\epsilon^2)$ [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 parameters, our primary goal is to understand the complexity of decoding $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-locally-list-decodable codes that have polynomial rate (i.e. where $N(M) = \mathrm{poly}(M)$).

Our main result characterizes the complexity of local-list-decoders for such codes. We show that computing majority on $\Theta(1/\epsilon)$ bits is essentially equivalent to $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-local-list-decoding: 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 $\Theta(1/\epsilon)$ bits. In the other direction, there is an explicit $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-locally-list-decodable code with a very efficient (in terms of size and depth) local-decoder that uses majority gates of fan-in $\Theta(1/\epsilon)$. This is stated (informally) in the following theorem.

Theorem 1 (Informal). *If there exists a binary code with a $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-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 $\Theta(1/\epsilon)$ bits.*

In the other direction, there exist (for every $\epsilon \geq 1/2^{\sqrt{\log(M)}}$) explicit binary codes of polynomial rate, with a $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-local-list-decoder. The decoder is a constant depth circuit of size $\mathrm{poly}(\log M, 1/\epsilon)$ with majority gates of fan-in $\Theta(1/\epsilon)$.

The upper bound follows by replacing one of the ingredients in the construction of Goldwasser et. al. [GGH$^+$07], with a modification of the recent de-randomized direct-product construction of Impagliazzo *et al.* [IJKW08], thus improving the code's rate. Our main technical contribution is in the lower bound, where we show a reduction from computing majority over inputs of size $\Omega(1/\epsilon)$ to local-list-decoding binary codes with good parameters. In fact, our lower bound holds for any $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-list-decodable binary code, regardless of its rate. By known lower bounds on the size of constant-depth circuits that compute majority [Raz87, Smo87], we obtain the following corollary.

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

In particular, this result shows that the noise rate from which the constant-depth local-list-decoders of [GGH$^+$07] recover is essentially optimal. And thus we get an exact characterization of what is possible with constant-depth decoders: up to radius $1/2 - 1/\mathrm{poly}\log\log N$ 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 $(1/2-\epsilon, \ell)$-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 $O(1/\epsilon)$ bits. This means that even if the list size is *sub-exponential* in $1/\epsilon$, the size of the decoder still must be nearly exponential in $1/\epsilon$ (even if the decoder is allowed $\bmod q$ 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 [STV99, TV02, Tre03, Vio03], it is well known that there is a tight connection between binary locally (list) decodable codes and hardness amplification. Using this connection, we obtain limits (in the spirit of Corollary 1) on (black-box) hardness amplification procedures. We defer the statement of these results and a discussion to Section 5.

1.2 Related Work

Positive Results. Goldreich and Levin [GL89] were the first to (implicitly) consider locally-list-decodable codes. They showed that the Hadamard code has such a decoder. The first locally-list-decodable code with "good" parameters was obtained by Sudan, Trevisan and Vadhan [STV99].[3] They showed that the Reed-Muller code concatenated with the Hadamard code has such a decoder. Their decoder is in the class NC^2. That is, the bound on the circuit's size is $\mathrm{poly}(\log M, 1/\epsilon)$, while the bound on the depth is $O(\log^2(\log M))$ (i.e. the square of the logarithm of its input's size). Goldwasser *et. al.* [GGH$^+$07] construct binary codes that are $(1/2 - \epsilon, \mathrm{poly}(1/\epsilon))$-locally-list decodable by constant depth circuits of size $\mathrm{poly}(\log M, 1/\epsilon)$ with majority gates of fan-in $O(1/\epsilon)$. The rate of these codes is exponential in $1/\epsilon$. In particular, for $\epsilon \geq 1/\log\log M$ their codes have constant depth decoders of size $\mathrm{poly}\log M$ that do not use any majority gates. This is simply because for such large enough ϵ the fan-in of the majority gates is so small relative to the circuit size that majority computations can be done in constant depth (this only increases the decoder size by a polynomial factor).

Lower Bounds. One of the main problems that was left open in [GGH$^+$07], and the main motivation for our work, is whether the parameters of their codes, especially the noise rate from which local-list-decoding is possible, can be improved, while maintaining the size and depth of the decoder.[4] In this work we show that while the rate of these codes can be improved, the list-decoding radius of $1/2 - 1/\log\log M$ cannot (for small constant depth decoders *without* majority gates). The lower bound on the list-decoding radius follows from Corollary 1.

The question of lower bounding the complexity of local-list-decoders was raised by Viola [Vio06]. He conjectured that locally $(1/2 - \epsilon, \ell)$-list-decodable codes require computing majority over $O(1/\epsilon)$ bits,[5] even when the list size ℓ is exponential in $1/\epsilon$. 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 5). 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:

Known Results for *Non-Local* Decoders. 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 $1/2 - \epsilon$ can be used to compute majority on $\Theta(1/\epsilon)$ 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 $\mathrm{poly}(1/\epsilon)$. 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

[3]The Hadamard code does not have "good" parameters since its codewords are of length 2^M.

[4]Significant improvements in the list-decoding radius would have important implications to the construction of pseudorandom generators (see Section 5).

[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 $O(1/\epsilon)$ bits.

have consequences for fairly large values of ϵ. For example, when $\epsilon \geq 1/\log N$, the only implication that we get from [Vio06], is that there is a constant-depth circuit of size at least $N \geq 2^{1/\epsilon}$ that computes majority on instances of size $1/\epsilon$. But this is trivially true, and thus we do not get any contradiction. In the local-decoding setting the decoders' circuits are much smaller and thus we can obtain limitations for much larger ϵ's. In this paper we rule out constant-depth decoders for $(1/2-\epsilon, \mathrm{poly}(1/\epsilon))$-local-list-decoders for any ϵ smaller than $1/\mathrm{poly}\log\log N$ (recall that this matches the construction of [GGH+07]).

Known Results for *Specific* Codes. 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).

Known Results for *Non-Adaptive* Decoders. Recently (simultaneously and independently of our work), Shaltiel and Viola [SV08] gave a beautiful proof of the conjecture for the local-decoding setting, with ℓ exponential in $1/\epsilon$, 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 $1/\epsilon$, 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.

1.3 On the Choice of Parameters

In this work 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. [Yek07]). Furthermore, there are negative results that suggest that local-decoding with small query complexity may *require* large rate [KT00, KdW04, 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 $m \in \{0,1\}^*$ we denote by $m[i]$ the i'th bit of m. $[n]$ denotes the set $\{1,\ldots,n\}$. For a finite set S we denote by $x \in_R S$ that x is a sample uniformly chosen from S. 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 Complexity Classes

Boolean circuits in this work always have NOT gates at the bottom and unbounded AND and OR gates. Such circuits may output more than one bit. Whenever we use circuits with gates that compute other functions, we explicitly state so. For a positive integer $i \geq 0$, AC^{i} circuits are Boolean circuits of size poly(n), depth $O(\log^i n)$, and unbounded fan-in AND and OR gates (where n is the length of the input). $AC^i[q]$ (for a prime q) are similar to AC^{i} circuits, but augmented with $\mod q$ gates. Throughout, we extensively use oracle circuits: circuits that have (unit cost) access to an oracle computing some function.

2.2 Locally list-decodable codes

Definition 1 (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 $(d(M), \ell(M))$-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 $\Delta_\Gamma(C_M(x), y) \leq d(M)$,*

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

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

Remark 1. *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 $\{C_a(\cdot,\cdot)\}_{a \in [\ell]}$, where $C_a(i, r_2) = D(a, i, r_1, r_2)$. In the second stage, it uses the advice a to pick the probabilistic circuit C_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).

[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.

2.3 Majority and related functions

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 $weight(x) = k-1\}$
$\Pi_{No} = \{x : x \in \{0,1\}^{2k}$ for some $k \in \mathbb{N}$ and $weight(x) = k\}$

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 1 ([Vio06])**.** *Let $\{C\}_{M\in\mathbb{N}}$ be a circuit family of size $S(M)$ and depth $d(M)$ for solving the promise problem Π on inputs of size M.*

Then, for every $M \in \mathbb{N}$, there exists a circuit B_M of size poly$(S(M))$ and depth $O(d(M))$ that computes majority on $2M$ 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.

3 Local-List-Decoding Requires Computing Majority

Theorem 2. *Let $\{C_M : \{0,1\}^M \to \{0,1\}^{N(M)}\}_{M\in\mathbb{N}}$ be a $(1/2-\epsilon(M), \ell(M))$-locally-list-decodable code, such that $\ell(M) \le 2^{\kappa\cdot M}$, and $1/N^{\delta_1} \le \epsilon \le \delta_2$ 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 poly$(S(M), \ell(M))$ and depth $O(d(M))$, that computes majority on $\Theta(1/\epsilon(M))$ 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 2. Fix a message length M, take $\epsilon = \epsilon(M)$. We will describe a circuit B with the stated parameters that decides the promise problem Π on inputs of length $1/\epsilon$. By Claim 1 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[8] 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 $1/2\epsilon$. 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 $1/2 - 2\epsilon$, and the decoding algorithm successfully recovers every bit of m w.h.p. 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

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

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 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 $1/2 - 2\epsilon$, 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 $1/2 - 2\epsilon$ (and error-rate $1/2$ above it), over the case that up to the $(k-1)$'th level the error-rate is $1/2 - 2\epsilon$ (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 $1/2 - 2\epsilon$), 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 $C(m)$). At first glance it is not clear that we have gained anything, because we still have to provide answers for queries above the k'th level 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 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 $O(\log(d))$.

Proof of Theorem 2. Fix $M \in \mathbb{N}$, $C = C_M : \{0,1\}^M \to \{0,1\}^{N(M)}$, $\epsilon = \epsilon(M)$, $\ell = \ell(M)$, $S = S(M)$ 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 $1/\epsilon$ (and thus also for computing majority, by Claim 1) 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 1). 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:

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

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 $1/2 - 2\epsilon$ 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 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 $1/2 - 2\epsilon$ 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 3. *For every message $m \in \{0,1\}^M$:*

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

Proof. Recall that the decoder D has the guarantee that if a codeword is corrupted in less than a $1/2 - \epsilon$-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 $1/2 - \epsilon$-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 $1/2 - 1/2\epsilon$. Then the probability that the fraction of coordinates corrupted is more than $1/2 - 1/\epsilon$ is exponentially small in $1/\epsilon$ (here we use that fact that $1/\epsilon$ is significantly smaller than N, because $\epsilon \geq 1/N^{\delta_1}$ 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 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 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 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 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 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,

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

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 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 $1/2 - 2\epsilon$ and for the last $d-k$ levels we give it access to $C(m)$ corrupted with error rate $1/2$. By (2):

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

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,

$$\Pr_{\bar{r}, E_0, \ldots, E_0, E_1, \ldots, E_1}[A_2^{O^k}(\bar{r}) = 1] - \Pr_{\bar{r}, E_0, \ldots, E_0, E_1, \ldots, E_1}[A_2^{O^{k-1}}(\bar{r}) = 1] \geq \frac{1}{2d} \tag{3}$$

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 (3) 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 poly(S, ℓ) 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 (3)). Let $\bar{r}'$ be the randomness of A. We have,

$$\Pr_{e \leftarrow E_1, \bar{r}'}[A^e(\bar{r}') = 1] - \Pr_{e \leftarrow E_0, \bar{r}'}[A^e(\bar{r}') = 1] \geq \frac{1}{2d} \tag{4}$$

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 $1/2\epsilon$ with a small advantage (that will later be amplified). We assume w.l.o.g. that $1/2\epsilon$ is an even integer. On input $x \in \Pi_{yes} \cup \Pi_{No}$ of length $1/2\epsilon$, B runs A while simulating the noise $e \leftarrow E_b$ as follows: whenever A queries e in position j, B chooses uniformly $i \in [1/2\epsilon]$ 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 $\mathrm{poly}(S, \ell)$ and depth $O(d)$ (inherited from A). If $x \in \Pi_{Yes}$, then $\Pr_i[x[i] = 1] = 1/2 - 2\epsilon$, 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 (4):

Claim 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 $\mathrm{poly}(d)$ 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 and a Chernoff bound, this amplified version of B computes Π correctly on instances of size $1/2\epsilon$ with probability of error at most $1/10$. Furthermore, it is a circuit of size $\mathrm{poly}(S, \ell)$ and depth $O(d)$ (note that counting the number of 1-answers in the $\mathrm{poly}(d)$ executions that are run for the final amplification step only requires additional depth $O(\log(d))$).

$\square$

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 2. *Let $\{C_M : \{0,1\}^M \to \{0,1\}^{N(M)}\}_{M \in \mathbb{N}}$ be a $(1/2 - \epsilon, \ell)$-locally-list-decodable code (where ϵ is in the range specified in Theorem 2) 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 $s = 2^{(1/\epsilon)^{\Omega(1/d)}}/poly(\ell)$.*

4 Majority Suffices for Local-List-Decoding

Theorem 3. *For every $2^{-\Theta(\sqrt{\log M})} \leq \epsilon = \epsilon(M) < 1/2$, there exists a $(1/2 - \epsilon, poly(1/\epsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \{0,1\}^{poly(M)}\}_{M \in \mathbb{N}}$ with a local-decoder that can be implemented by a family of constant depth circuits of size $poly(log M, 1/\epsilon)$ using majority gates of fan-in $\Theta(1/\epsilon)$ (and AND/OR gates of unbounded fan-in).*

Remark 2. *The construction above only applies for $\epsilon \geq 2^{-\Theta(\sqrt{\log M})}$. Thus we fall slightly short of covering the whole possible range (since one can hope to get such codes for $\epsilon = 1/M^\delta$ for a small constant δ). We note, however, that the range of ϵ which is most interesting for us is between $1/poly \log M$ and $1/poly \log \log M$ (see the discussion in the introduction) which we do cover. We also mention that if one insists on codes with $\epsilon = 1/M^\delta$, then we can construct such codes with quasi-polynomial rate (below we state without proof the exact parameters of these codes).*

To prove Theorem 3, we concatenate three codes. The first, by [GGH+07], is a binary locally-decodable code that can be uniquely decoded from a constant relative distance.

Theorem 4 ([GGH+07])**.** *There is an explicit code $\{C_M : \{0,1\}^M \to \{0,1\}^{poly(M)}\}_{M \in \mathbb{N}}$ that can be locally decoded (uniquely, i.e. with list size 1) from distance $1/25$ by probabilistic AC^0 circuits of size $poly(\log M)$.*

The second code that we need is a non-binary approximate locally-list-decodable code. This code (as well as its decoder) is a modification of the de-randomized direct product code of [IJKW08]. We now describe this code, starting by defining the notion of approximate codes.

Definition 2. *[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 1 with the following relaxation of (1):*

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

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 1). In our description of the decoders, we will use the two-stage process view of this definition as discussed in Remark 1.

We can now state and prove the existence of the approximate codes that we need, these are a modification of the codes of [IJKW08], that allows local decoding in AC^0.

Theorem 5. *For every $\delta = O(1)$ and every $2^{-\Theta(\sqrt{\log M})} \leq \epsilon = \epsilon(M) < \delta$, there exists a δ-approximate $(\epsilon, poly(1/\epsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \Gamma^{poly(M)}\}_{M\in\mathbb{N}}$. Where $|\Gamma| = poly(1/\epsilon)$. The code has a local decoder that can be implemented by constant depth circuits of size $poly(logM, 1/\epsilon)$.*

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

In terms of complexity, the decoder for C_{in} can be implemented in AC^0 (this is shown in [IJKW08]). We do not, however, know how to implement the decoder for C_{out} in AC^0. Therefore, to get a code decodable in AC^0, we modify their outer code (we abuse notation and still call the modified code C_{out}) and present an 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 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 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 $\mathbb{F}_q$, where $q^m = M$, and $q^8 = k = \log|\Sigma| = \text{poly}(1/\epsilon)$. The codeword is indexed by tuples of 9 vectors in $\mathbb{F}_q^m$, and thus each codeword is of length $O(|\mathbb{F}_q^m|^9) = O(M^9) = \text{poly}(M)$.

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. $\vec{v_1}, \ldots, \vec{v_8}, \vec{s} \in \mathbb{F}_q^m$). Consider the affine subspace B spanned by $\vec{v_1}, \ldots, \vec{v_8}$ and shifted by $\vec{s}$

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

Recalling that entries in the message are associated with vectors in $\mathbb{F}_q^m$ (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 $q^8 = \text{poly}(1/\epsilon)$, and thus the alphabet size is exponential in $1/\epsilon$, as claimed above. We think of each (9-vector) index in the codeword as an 8-dimensional affine subspace of $\mathbb{F}_q^m$, 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 6. *Let $\vec{v_1},\ldots,\vec{v_8}$ be chosen uniformly and independently from $\mathbb{F}_q^m$. 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, $1/\sqrt{M} << \epsilon$. 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 1). Given access to a string y that has Hamming distance at most ϵ from some codeword $C_{out}(msg)$, D produces $O(1/\epsilon^2)$ 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 $\vec{v_1},\ldots,\vec{v_8},\vec{s} \in \mathbb{F}_q^m$. By Claim 6, 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 $\mathbb{F}_q^m$). 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 $O(\frac{\log(1/\delta)}{\epsilon})$ 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 AC^0 circuits of size $\mathrm{poly}(q) = \mathrm{poly}(1/\epsilon)$.

Claim 7. *Given an 8-dimensional affine subspace $B \subseteq \mathbb{F}_q^m$, represented by $(\vec{v_1}, \dots, \vec{v_8}, \vec{s})$. A probabilistic AC^0 circuit of size poly$(\log M, 1/\epsilon)$ 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 $(\vec{w_1}, \dots, \vec{w_4})$ in B, by taking random linear combinations of $\vec{v_1}, \dots, \vec{v_8}$. With high probability (more than say 1/2), these vectors will be linearly independent. This can be verified in AC^0 by enumerating all $q^4 = \mathrm{poly}(1/\epsilon)$ linear combinations, and thus the success probability can be amplified (in parallel time) to $1 - 1/\mathrm{poly}(M, 1/\epsilon)$. 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 $(\vec{w_1}, \dots, \vec{w_4})$ and shifted by $\vec{t}$. All of these operations can be done by an AC^0 circuit of size $\mathrm{poly}(\log M, 1/\epsilon)$.

□

Claim 8. *Given a 4-dimensional affine subspace $A \subseteq \mathbb{F}_q^m$, represented by $(\vec{v_1}, \dots, \vec{v_4}, \vec{s})$ and a vector $\vec{j} \in \mathbb{F}_q^m$. An AC^0 circuit of size $O(\log 1/\epsilon)$ can check whether $\vec{j} \in A$.*

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

□

Claim 9. *Given a 4-dimensional affine subspace $A \subseteq \mathbb{F}_q^m$, represented by $(\vec{v_1}, \dots, \vec{v_4}, \vec{s})$ and a vector $\vec{j} \in \mathbb{F}_q^m \setminus A$, a probabilistic AC^0 circuit of size poly$(\log M, 1/\epsilon)$ can sample vectors $(\vec{w_1}, \dots, \vec{w_8}, \vec{t})$ 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 $(\vec{v_1}, \dots, \vec{v_4}, \vec{j} - \vec{s})$ (as above, this can be done in 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 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 6. *For every $0 < \epsilon(m) < 1/2$, The Hadamard code, $Had : \{0,1\}^m \to \{0,1\}^{2^m}$ is a $(1/2 - \epsilon, \frac{1}{\epsilon^2})$-local-list-decodable-code. The decoder can be implemented by an AC^0 circuit of size poly$(m, 1/\epsilon)$ that uses majority gates of fan-in $\Theta(1/\epsilon)$.*

We can now put everything together and prove Theorem 3.

Proof of Theorem 3. Fix $\epsilon(M) = \epsilon$ in the specified range. Our code C is a combination of the codes in Theorem 4 (we denote it here by C_1), the code in Theorem 5 (we denote it here by C_2) with $\epsilon_2 = \epsilon^3/2$ and $\delta = 1/25$, and the code in Theorem 6 (C_3), with $\epsilon_3 = \epsilon/2$. 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 poly$(1/\epsilon)$). 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 $N_3 = 2^k = \text{poly}(1/\epsilon) \leq \text{poly}(M)$, which is 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 1. *$C' : \{0,1\}^{N_1} \to \{0,1\}^N$ is a 1/25-approximate $(1/2 - \epsilon, \ell)$-locally-list-decodable code, where $\ell =$ poly$(1/\epsilon)$. The local decoder for C' can be implemented by constant-depth circuits of size poly$(\log M, 1/\epsilon)$, with majority gates of fan-in $O(1/\epsilon)$ (and AND/OR gates of unbounded fan-in).*

Proof. By Theorem 5 (and the simplification assumption from Remark 1), there is a locally-list-decoder D_2 taking advice of size $\log(\ell_2)$, where $\ell_2 = \text{poly}(1/\epsilon)$, such that for every $y \in \Gamma^{N_2}$ and for every $m \in \{0,1\}^{N_1}$ for which $\Delta_\Gamma(C_2(m), y) \leq 1 - \epsilon^3/2$,

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

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

By Theorem 6, there is a locally-list-decoder D_3 taking advice of size $\log(\ell_3)$, where $\ell_3 = O(1/\epsilon^2)$, such that for every $y \in \{0,1\}^{N_3}$ and for every $m \in \{0,1\}^k$ for which $\Delta(C_3(m), y) \leq 1/2 - \epsilon/2$,

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

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

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 $\Delta(C'(x), y) \leq 1/2 - \epsilon$, there are at least $\epsilon/2$ symbols of

$C_2(x)$ for which their C_3 encoding has $1/2+\epsilon/2$ 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 $\epsilon/2 \cdot \epsilon^2 = \epsilon^3/2$ 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 $\epsilon^3/2$, 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 $\text{poly}(\log M, 1/\epsilon)$. Both are of constant depth where the latter uses majority gates of fan-in $\Theta(1/\epsilon)$. Combining the two we get a constant-depth $(1/2-\epsilon, \text{poly}(1/\epsilon))$-locally-list-decoder for the concatenated code of size $\text{poly}(\log M, 1/\epsilon)$ with majority gates of fan-in $\Theta(1/\epsilon)$. □

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 $1/2-\epsilon$ 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 $\text{poly}(\log M, 1/\epsilon)$, and their depth is constant, then so are the size and depth of the combined decoder, and it uses majority gates of fan-in $\Theta(1/\epsilon)$ because so does the decoder D_2. □

As mentioned in Remark 2, we can obtain codes with quasi-polynomial rate that work for $\epsilon = 1/M^\delta$. These are obtained by replacing the code C'_M in the proof of Theorem 5, 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 7. *For every $1/M^\delta \leq \epsilon = \epsilon(M) < 1/2$ (where $\delta > 0$ is a constant), there exist a $(1/2-\epsilon, \text{poly}(1/\epsilon))$-locally-list-decodable code $\{C_M : \{0,1\}^M \to \{0,1\}^{M^{O(\log(1/\epsilon))}}\}_{M\in\mathbb{N}}$ with a local-decoder that can be implemented by a family of constant depth circuits of size poly$(logM, 1/\epsilon)$ that use majority gates of fan-in $\Theta(1/\epsilon)$ (and AND gates of unbounded fan-in).*

5 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 3. *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 3. 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 $1/2 + 1/\text{poly}n$) 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.

Our main result in this section, Theorem 8 below, shows that a function f that is hard enough to lead to de-randomizations *cannot be obtained via uniform (or even somewhat non-uniform) fully-black-box worst-case to average-case reductions.*

Our Results. 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 1. *There is a $(1/2 - \epsilon, \ell)$-locally-list-decodable code Enc : $\{0,1\}^K \to \{0,1\}^N$ with a decoder D, if and only if there is a $(2^{-k}, 1/2-\epsilon)$-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.*

Using this connection together with Theorem 2 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 8. *If there is a $(2^{-k}, 1/2-\epsilon(k))$-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 poly$(s(k), 2^{a(k)})$ and depth $O(d(k))$, that computes majority on $O(1/\epsilon(k))$ bits.*

It is known [Raz87, Smo87] that low complexity classes *cannot* compute majority. Thus, Theorem 8 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. 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 8 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 $O(\log 1/\epsilon)$-non-uniform) $(2^{-k}, 1/2 - \epsilon)$-fully-black-box worst-case to average-case reductions for ϵ smaller than $1/\text{poly}\log k$, where *Dec* is a $AC^0[q]$ circuit (for a prime q) of size poly$(k, 1/\epsilon)$. This should be contrasted with [GGH$^+$07] who showed such a fully-black-box reduction (with *Dec* in AC^0) for $\epsilon \geq 1/\log^{\beta} k$, 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 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 8 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 1 can be extended to show a similar equivalence between δ-approximate locally $(1/2 - \epsilon, \ell)$-list-decodable codes to $(\delta, 1/2 - \epsilon)$-fully-black-box hardness amplification (with the same translations between the parameters). Let $0 < \alpha < 1/2$ be an arbitrary constant. Theorem 2 can be extended to show that a $1/2 - \alpha$-approximate locally $(1/2 - \epsilon, \ell)$-list-decodable code implies circuits for majority with the same parameters as in the statement of Theorem 8.[9] Putting the two together we obtain the following.

Theorem 9. *Let $0 < \alpha < 1/2$ be an arbitrary constant. If there is a $(1/2 - \alpha, 1/2 - \epsilon(k))$-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 poly$(s(k), 2^{a(k)})$ and depth $O(d(k))$, that computes majority on $1/\epsilon(k)$ bits.*

We conclude with an informal discussion about *not necessarily fully* black-box hardness amplification. Note that in definition 3, the hardness amplification is required to work for every function f. A more relaxed notion is (not necessarily fully) black-box reductions:

Definition 4. *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

[9]The proof follows the outline of the proof of Theorem 2 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 $1/2 - \epsilon$ we can recover at least $1/2 + \alpha$ of the bits. So by sampling bits from m we can distinguish between the two cases.

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 8 and 9 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) $(\delta + 1/\text{poly}\log(k), 1/2 - \epsilon(k))$-hardness amplification from f to *any* other 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 $O(\epsilon(k))$-bit inputs.

The basic idea (very informally) is similar to the proof of Theorem 8. 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 AC^0 circuit, and we get a circuit for computing f. On the other hand, the reduction *does* recover from error rate $1/2 - \epsilon(k)$, computing f correctly with probability $1 - \delta - \text{poly}\log(k)$. This gives a distinguisher between error rates 1/2 and $1/2 - \epsilon(k)$, which in turn (as in the proof of Theorem 8) leads to an algorithm for computing majority on $O(\epsilon(k))$ bits. The full details are omitted from this extended abstract.

6 Acknowledgements

We thank Shafi Goldwasser and Salil Vadhan for their assistance and insightful comments. Thanks to Russell Impagliazzo for helpful discussions and for supplying us with a 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 a manuscript of [SV08].

References

[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.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

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

[GGH+07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 440–449, 2007.

[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] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

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

[IJKW08] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. Uniform direct-product theorems: Simplified, optimized, and derandomized. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008.

[KdW04] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69(3):395–420, 2004.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 80–86, 2000.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[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.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SV08] Ronen Shaltiel and Emanuele Viola. Hardness amplification proofs require majority. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008.

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

[TV02] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 129–138, 2002.

[TV07] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):361–364, 2007.

[Vio03] E. Viola. Hardness vs. randomness within alternating time. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 53–62, 2003.

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

[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.

[Yek07] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 266–274, 2007.