Approximate List-Decoding of Direct Product Codes and Uniform Hardness Amplification

Russell Impagliazzo
University of California, San Diego
La Jolla, CA, USA
russell@cs.ucsd.edu

Ragesh Jaiswal
University of California, San Diego
La Jolla, CA, USA
rjaiswal@cs.ucsd.edu

Valentine Kabanets
Simon Fraser University
Vancouver, BC, Canada
kabanets@cs.sfu.ca

Abstract

Given a message \( \text{msg} \in \{0,1\}^N \), its \( k \)-wise direct product encoding is the sequence of \( k \)-tuples \((\text{msg}(i_1), \ldots, \text{msg}(i_k))\) over all possible \( k \)-tuples of indices \((i_1, \ldots, i_k) \in \{1, \ldots, N\}^k\). We give an efficient randomized algorithm for approximate local list-decoding of direct product codes. That is, given oracle access to a word which agrees with a \( k \)-wise direct product encoding of some message \( \text{msg} \in \{0,1\}^N \) in at least \( \varepsilon \geq \text{poly}(1/k) \) fraction of positions, our algorithm outputs a list of \( \text{poly}(1/\varepsilon) \) strings that contains at least one string \( \text{msg}' \) which is equal to \( \text{msg} \) in all but at most \( k - \Omega(1) \) fraction of positions. The decoding is local in that our algorithm outputs a list of Boolean circuits so that the \( j \)th bit of the \( i \)th output string can be computed by running the \( i \)th circuit on input \( j \). The running time of the algorithm is polynomial in \( \log N \) and \( 1/\varepsilon \). In general, when \( \varepsilon > e^{-k^\alpha} \) for a sufficiently small constant \( \alpha > 0 \), we get a randomized approximate list-decoding algorithm that runs in time quasipolynomial in \( 1/\varepsilon \), i.e., \( (1/\varepsilon)\text{poly log } 1/\varepsilon \).

As an application of our decoding algorithm, we get uniform hardness amplification for \( \mathsf{P}^{\mathsf{NP}} \), the class of languages reducible to \( \mathsf{NP} \) through one round of parallel oracle queries: If there is a language in \( \mathsf{P}^{\mathsf{NP}} \) that cannot be decided by any \( \mathsf{BPP} \) algorithm on more that \( 1 - 1/n^{\Omega(1)} \) fraction of inputs, then there is another language in \( \mathsf{P}^{\mathsf{NP}} \) that cannot be decided by any \( \mathsf{BPP} \) algorithm on more that \( 1/2 + 1/n^{\omega(1)} \) fraction of inputs.

1. Introduction

There is a rich interplay between coding theory and computational complexity. Complexity has both benefited from and contributed to coding theory. For instance, the PCP Theorem [AS98, ALM+98] uses error-correcting codes based on polynomials over finite fields, while the final construction gives rise to a new kind of error-correcting code, a locally testable encoding of satisfying assignments of propositional formulas. In derandomization, error-correcting codes are (explicitly or implicitly) behind many constructions of pseudorandom generators from hard Boolean functions [NW94, BFNW93, IW97, STV01, SU01, Uma03]. The breakthrough extractor
construction of Trevisan [Tre01] combines good list-decodable error-correcting codes and the pseudorandom generator of Nisan and Wigderson [NW94]. (For many other connections between coding and complexity, see the excellent survey [Tre04].)

Another connection between coding and complexity was observed in [Tre03, Imp02], who show that direct product lemmas yield locally, approximately list-decodable codes. Direct product lemmas (e.g., Yao’s XOR Lemma [Yao82]) are formalizations of the intuition that it is harder to compute a function on many independent instances than on a single instance. In such lemmas, a function $f$ that is hard to compute on some $\delta$ fraction of inputs is used to construct a function $\hat{f}$ that is hard to compute on a larger fraction (usually written as $1/2 - \epsilon$) of inputs. View $\hat{f}$ as a “coded” version of the “message” $f$. In an XOR lemma, it is shown how to construct a list of circuits containing a circuit that computes $f$ with fewer than $\delta$ fraction of errors from a circuit with fewer than $1/2 - \epsilon$ fraction of errors for $\hat{f}$. This corresponds to approximately list-decoding the code in that instead of exactly recovering the message $f$, the decoding algorithm finds a list of strings containing a string of small relative Hamming distance from $f$; moreover, the decoding is local since the constructed circuit computes an individual bit of the decoded message, not the entire message.

Approximately decodable codes are a relatively recent and potentially powerful tool in coding theory. For example, an approximately decodable code can be composed with a standard error-correcting code to boost the amount of noise that can be tolerated (see [ABN+92, GI01, GI02, GI03] for several applications of this idea.) Weakening the notion of error-correction to include approximate error-correcting codes dramatically increases the available options for coding functions. For example, the code implicit in the XOR lemma is the $k$-truncated Hadamard code, consisting of the inner products of the message string with just those strings of Hamming weight $k$ (where typically $k \ll N$, for $N$ the length of the message). This code has very small sensitivity: flipping one bit of the input changes only a small portion of output bits. On the other hand, any true error-correcting code has large distance and hence large sensitivity. This difference translates to the fact that in case of truncated Hadamard code, given a corrupted codeword of some message $msg$, it is impossible to recover a short list of strings that would contain $msg$, though it is possible to do the same for Hadamard code. On the other hand, for truncated Hadamard code, it is possible to recover a short list of messages such that at least one of them is close to $msg$ (see the Appendix).

From a combinatorial viewpoint, this means that approximately decodable codes escape many of the restrictions that have been proved for standard error-correction. In complexity terms, this allows the constructed function $\hat{f}$ to be locally computable from $f$, $\hat{f} \in P_f$, an important property if we want the constructed function to have a similar complexity to the original.

As Trevisan [Tre03] details, the standard proofs of the XOR lemma give an approximate list-decoding algorithm for the $k$-truncated Hadamard code running in time (and producing a list of size) exponential in $\text{poly}(1/\epsilon)$, when given a circuit with $\epsilon$ correlation with the codeword. He observes that the list size and hence time for such an algorithm is exponential in the amount of advice the construction uses. In the present paper, we reduce the amount of advice in a proof of the XOR Lemma, giving an approximate list-decoding algorithm with polynomial time and list length for $\epsilon > 1/\text{poly}(k)$ for any polynomial $\text{poly}$, at the price of a somewhat weaker approximation ratio $\delta$. As a consequence, we get a strong hardness amplification result for $P^\text{NP}_{\parallel}$, the class of problems reducible to NP through one round of parallel oracle queries.

While interesting in themselves, our results are also significant in that they utilize the equivalence of XOR lemmas and approximate decoding at several points. As in many other cases, knowing an equivalence between problems in two different domains gives researchers leverage to make progress in both domains.

There are three main ideas used in the construction:
Self-advising direct products: In a proof of a direct product theorem, one is converting a circuit that solves many instances of a function a small fraction of the time into one that solves a single instance a large fraction of the time. In the non-uniform setting, the converting algorithm is allowed access to many bits of advice, which in most proofs consists of random solved instances of the problem. We use the circuit solving many instances of the function on a random tuple of instances, and hope that it provides correct answers for those instances. If so, we use many of these answers in place of advice. We use a sampling lemma to show that these are still almost uniform, even conditioned on the success of the direct product circuit. Unfortunately, while this provides some of the required advice, it is usually not enough.

Direct product boosting: To increase the amount of advice yielded by the method above, we show how to boost a direct product circuit, constructing a circuit that approximately solves a larger direct product from one that (approximately) solves a smaller direct product. We apply this method recursively to stretch the direct product by any polynomial amount. Thus, we get a circuit that solves almost all of a larger number of instances a small fraction of the time, from one that solves all of a smaller number of instances. This can be viewed as the special case of approximately list decoding the truncated Hadamard code, when the message size and $k$ are polynomially related.

Fault tolerant direct products: Combining the two ideas above, we can generate a large amount of slightly flawed advice, in that the instances are close to uniform and most of the supposed function values are correct. We show that at least one of the standard proofs of the direct product theorem, the one from [IW97] can tolerate such faulty advice.

Below, we describe our results and techniques more precisely.

1.1 Direct Product Lemma and Direct Product Codes

Given a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$, we define its $k$-wise direct product as $f^k(x_1, \ldots, x_k) = f(x_1) \cdots f(x_k)$. The Direct Product Lemma [Yao82, Lev87, Imp95, GNW95, IW97] says that if a Boolean function $f$ cannot be computed by any size $s$ Boolean circuit on more than $1 - \delta$ fraction of inputs, then its direct product function $f^k$ cannot be computed by any circuit of size $s' = s \cdot \text{poly}(\epsilon, \delta)$ on more than $\epsilon = e^{-\Omega(\delta k)}$ fraction of inputs. Viewing $f^k$ as a “direct product” encoding of the message $f$, we can interpret the (contrapositive to the) Direct Product Lemma as a statement about approximate list-decodability of this direct-product code.

The known proofs of the Direct Product Lemma [Imp95, GNW95, IW97] are constructive: there is an algorithm that, given a circuit of size $s'$ that computes $f^k$ on more than $\epsilon$ fraction of inputs, constructs a Boolean circuit of size $s$ that computes $f$ on more than $1 - \delta$ fraction of inputs. However, in all these proofs, the algorithm constructing a circuit for $f$ needs some nonuniform advice. Instead of producing a circuit for $f$, the algorithm actually produces a list of circuits one of which computes $f$ on more than $1 - \delta$ fraction of inputs; the logarithm of the size of this list is exactly the number of bits of nonuniform advice used by the construction. In the known proofs, the advice size is $\text{poly}(1/\epsilon)$ (cf. [Tre03]) and so the size of the list of circuits is $2^{\text{poly}(1/\epsilon)}$. In terms of approximate list-decoding, this is the list of possible approximate messages, one of which is $\delta$-close to the original message. For information-theoretically optimal approximate list decoders, this list size does not exceed $O(1/\epsilon^2)$ (see the Appendix), which corresponds to $O(\log(1/\epsilon))$ bits of advice.

In this paper, we achieve list size $\text{poly}(1/\epsilon)$, albeit only for large $\epsilon = \Omega(\text{poly}(1/k))$. To state our main theorem, we need the following definition. We say that a circuit $C \epsilon$-computes the
Boosting direct products

We give an algorithm for converting a circuit that computes a direct product, where one has \( \gamma = 0 \). In our case, \( C \) is only required to get most of the answers right. However, since we use this weaker notion recursively inside our main algorithm, and since it gives a stronger result, we state our main theorem for this notion of computing the direct product.

**Theorem 1 (Main Theorem).** There is a randomized algorithm \( A \) with the following property. Let \( f \) be any \( n \)-variable Boolean function, and let \( C \) be a circuit that \( \epsilon \)-computes \((1 - k^{-\mu})\) Direct Product \( f^k \), where \( \mu > 0 \) and \( \epsilon > e^{-k^\alpha} \) for a sufficiently small constant \( \alpha \) dependent on \( \mu \) (e.g., \( \alpha = \min\{\mu/1000, 0.0001\} \)). Given circuit \( C \), algorithm \( A \) outputs with probability at least \( \epsilon' = \epsilon^{\text{poly} \log k / \epsilon} \) a circuit \( C' \) such that \( C' \) agrees with \( f \) on at least \( 1 - \rho \) fraction of inputs, where \( \rho = O(k^{-\nu}) \) for some constant \( \nu > 0 \) dependent on \( \mu \) (with \( \nu \approx \min\{\mu/2, 0.2\} \)). The running time of the algorithm \( A \) is at most \((|C|/\epsilon)^{\text{poly} \log 1/\epsilon}, \) and the size of \( C' \) is at most \( \text{poly}(|C|/\epsilon) \).

Note that the above theorem implies the list size at most \( \text{poly}(1/\epsilon') \), since by running the algorithm \( A \) for \( \text{poly}(1/\epsilon') \) times, we can construct a list of circuits one of which almost certainly is a \( 1 - \rho \) approximation to \( f \). In general, this list size will be at most \((1/\epsilon)^{\text{poly} \log 1/\epsilon}, \) i.e., \( \text{quasi-polynomial} \) in \( 1/\epsilon \). However, for the important special case where \( \epsilon > \text{poly}(1/k) \), the expression for \( \epsilon' \) above simplifies to \( \text{poly}(\epsilon) \), and so we get a list size \( \text{polynomial} \) in \( 1/\epsilon \).

Combining our local approximate list-decoding algorithms with the list-decoding algorithm for Hadamard codes due to Goldreich and Levin [GL89], we get local approximate list-decoding algorithms for truncated Hadamard codes, whose running time and list size are essentially those of the approximate direct-product decoding algorithms.

### 1.2 Our techniques

The proof of the Direct Product Lemma in [IW97] yields an efficient algorithm \( \text{LEARN} \) with the following property: Given as input a “small” circuit \( C \) computing the direct product function \( f^k \) for at least \( \epsilon \) fraction of inputs, and given about \((1/\epsilon^2)\) random samples of the form \((x, f(x))\) for independent uniformly distributed \( x \)s, the algorithm \( \text{LEARN} \) produces, with high probability, a “small” circuit computing \( f \) on at least \( 1 - \delta \) fraction of inputs, for \( \delta \approx \frac{\log(1/\epsilon)}{k} \). In our case, we have a circuit \( C \), but no labeled examples \((x, f(x))\).

Our construction combines three technical steps:

**Boosting direct products** We give an algorithm for converting a circuit that \( \epsilon \)-computes \((1 - \gamma)\) direct product \( f^k \) to one that \( \epsilon' \)-computes \((1 - \gamma')\) direct product \( f^{k'} \), where \( \epsilon' \geq \text{poly}(\epsilon) \), \( \gamma' \in O(\gamma + k^{-4}) \) and \( k' = k^{1.5} \). Repeating recursively, we can go from a circuit to compute a direct product on \( k \) inputs to one that approximately computes a direct product on any polynomial in \( k \) inputs. This direct product booster can be thought of as approximately list-decoding the \( k \)-truncated Hadamard code for the special case of “large” \( k \) where \( k \geq N^{\Omega(1)} \).

The main idea of the direct product booster is: given \( k^{1.5} \) inputs, first guess one subset \( S \) of \( k \) inputs and hope that the given circuit (approximately) computes the direct product on \( S \). Given that this first step succeeds, we use the values of \( f \) on inputs from \( S \) as a reality check on random subsets \( T \), accepting the values for inputs in \( T \) if there are few inconsistencies with the assumed values for \( S \). By the birthday paradox, \( S \) and \( T \) will have a large intersection, so if the values for \( S \) are (mostly) correct, we are unlikely to accept any \( T \) for which the values are not mostly correct. By combining many random consistent \( T \)’s, we eventually fill in correct guesses for most of the inputs in the entire set.
**Self-advising learning algorithm** The advice we need for LEARN is in the form of many random examples \((x, f(x))\). A circuit \(\epsilon\)-computing a direct product has an \(\epsilon\) chance of providing \(k\) such examples. To get enough samples, we first need to boost the direct product until \(k' = \text{poly}(1/\epsilon)\). However, the resulting samples may be correlated, and our circuit for \(k'\) only computes an approximate direct product. We quantify the first problem through a sampling lemma, which argues that a random subset of the inputs where the direct product circuit is (approximately) successful is almost uniform.

**Fault-tolerant learning algorithm** Finally, we address the last problem that some of the advice may in fact be misleading, not actually being the value of the function on the example input. To handle this, we give a fault-tolerant analysis of the learning algorithm from [IW97], showing that the algorithm works even if a small fraction of the advice is incorrect.

### 1.3 Uniform hardness amplification

The main application of the Direct Product Lemma (or Yao’s XOR Lemma) is to hardness amplification: If a Boolean function \(f\) is somewhat hard to compute on average, its XOR function \(f^{\oplus k}(x_1, \ldots, x_k) = \oplus_{i=1}^{k} f(x_i)\) is much harder on average. The known proofs of Yao’s XOR Lemma use nonuniform reductions and so they give hardness amplification only in the nonuniform setting of Boolean circuits.

Impagliazzo and Wigderson [IW01] consider the setting of uniform hardness amplification. Here one starts with a Boolean function family that is somewhat hard on average to compute by probabilistic polynomial-time algorithms, and defines a new Boolean function family that is much harder on average. Ideally, one would start from a function that is hard \(1/poly(n)\) of the time for some fixed polynomial \(\text{poly}(n)\), and end with a function in the same complexity class that is hard \(1/2 - 1/poly'(n)\) of the time, for any polynomial \(\text{poly}'(n)\). Yao’s XOR Lemma amplifies hardness of a Boolean function family \(f\) also in this setting, but only if we are given oracle access to \(f\). This oracle access can be eliminated under certain circumstances, e.g., if the distribution \((x, f(x))\) can be sampled, or if \(f\) is downward self-reducible and random self-reducible. Impagliazzo and Vadhan [TV02] show that uniform hardness amplification is also possible in PSPACE and in EXP.

Trevisan [Tre03, Tre05] considers uniform hardness amplification for languages in NP; the nonuniform case was studied in [O’D04, HVV04]. Trevisan [Tre05] shows uniform amplification from \(1/poly(n)\) to \(1/2 - 1/poly\log(n)\). Note that the final hardness falls short of the desired \(1/2 - 1/poly(n)\). The reason for this is the use of poly\((1/\epsilon)\)-bit advice by the BPP algorithm that, given a circuit computing an NP language \(L'\) on more than \(1/2 + \epsilon\) fraction of inputs, produces a circuit computing \(L\) on more than \(1 - 1/poly(n)\) fraction of inputs. If \(\epsilon = \log^{-\alpha} n\), for sufficiently small \(\alpha > 0\), then the required amount of advice is \(O(\log n)\). Using the average-case version of the “search-to-decision” reduction for NP [BDCGL92], this logarithmic advice can be eliminated in time \(2^{O(\log n)} = poly(n)\) by, essentially, trying all possible advice strings.

Using our efficient approximate list-decoding algorithm for truncated Hadamard codes, we achieve better uniform hardness amplification, but only for the class \(P^{\text{NP}_{su}}\). Namely, we prove the following.

**Theorem 2.** Suppose there is a Boolean function family \(f \in P^{\text{NP}_{su}}\) and a constant \(c\) such that \(f\) cannot be computed by any probabilistic polynomial-time algorithm on more than \(1 - 1/n^c\) fraction of inputs. Then there is a Boolean function family \(g \in P^{\text{NP}_{su}}\) that cannot be computed by any probabilistic polynomial-time algorithm on more than \(1/2 + 1/n^d\) fraction of inputs, for any constant \(d\).
The reason we get amplification for $P^{NP\parallel}$ rather than $NP$ is our use of the XOR function as an amplifier; if $f \in NP$, then its XOR function $f^{\oplus k}(x_1, \ldots, x_k) = \oplus_{i=1}^k f(x_i)$ is not necessarily in $NP$, although it is certainly in $P^{NP\parallel}$. (Achieving the same amplification for $NP$ seems to require a similar result with a monotone function replacing $\oplus^k$.)

Outline of the paper We give some preliminaries in Section 2. We describe the main tools used in our approximate list-decoding algorithm, and give the proof of our Main Theorem in Section 3. The auxiliary lemmas used in the proof of Main Theorem are proved in Section 4. Applications to uniform hardness amplification are given in Section 5. We give concluding remarks in Section 6. The appendix contains the information-theoretic upper bounds on the list size for optimal approximate list decoders of direct-product codes.

2 Preliminaries

2.1 Notation

For an integer $k > 0$, we will sometimes denote a set $\{1, \ldots, k\}$ by $[k]$. We use $\|v\|_1$ to denote the $\ell_1$-norm of a vector $v = (v_1, \ldots, v_n)$ where $\|v\|_1 = \sum_{i=1}^n |v_i|$.

2.2 Some definitions and theorems

Definition 3 (Statistical Distance). Given two distributions $D_1$ and $D_2$ over $\{0,1\}^n$, the statistical distance between them is defined as half of the $\ell_1$-norm of the vector $D_1 - D_2$, i.e.,

$$\text{Dist}(D_1, D_2) = \frac{1}{2} \cdot \sum_{x \in \{0,1\}^n} |D_1(x) - D_2(x)|,$$

where $D_i(x)$ denotes the probability of sampling $x$ from distribution $D_i$, for $i \in \{1,2\}$. Equivalently,

$$\text{Dist}(D_1, D_2) = \max_{T \subseteq \{0,1\}^n} |D_1(T) - D_2(T)|,$$

where $D_i(T)$ denotes the probability mass assigned by $D_i$ to the set $T$. For $\epsilon > 0$, we say that distributions $D_1$ and $D_2$ are statistically $\epsilon$-close if $\text{Dist}(D_1, D_2) \leq \epsilon$.

Definition 4 (Shannon Entropy). The Shannon Entropy of a distribution $D$ over $\{0,1\}^n$ is defined as

$$H(D) = - \sum_{x \in \{0,1\}^n} D(x) \log D(x).$$

We will use the following result from information theory which upperbounds the statistical distance between a distribution $P$ and the uniform distribution by the entropy deficiency of $P$; see, e.g., [CT91, Lemma 12.6.1].

Lemma 5. Let $P$ be any probability distribution over $\{0,1\}^n$ and let $U$ be the uniform distribution over $\{0,1\}^n$. Then

$$\|P - U\|_1^2 \leq (2 \log 2)(n - H(P)),$$

where $H$ is the Shannon entropy.
For a universe $S$ of size $N$ and a subset $T \subseteq S$ of size $m$, let $R$ be a uniformly random subset of $S$ of size $n$. The random variable $X = |R \cap T|$ is distributed according to the *hypergeometric distribution* with parameters $N$, $m$, and $n$. The expected value of $X$ is $nm/N$. The Hoeffding bound [Hoe63] says that this random variable is highly concentrated around its expectation. The proofs of the following versions of the Hoeffding bound can be found, for example, in [JLR00, Theorem 2.10].

**Theorem 6 (Hoeffding Bounds for Hypergeometric Distribution).** Let $X$ be a random variable which follows the hypergeometric distribution with parameters $N, m, n$. Let $\lambda = nm/N$ be the expectation of $X$. Then we have the following inequalities:

1. for $c > 1$ and $x \geq c\lambda$,
   \[
   \Pr[X \geq x] \leq e^{-c'x},
   \]
   where $c' = \log c - 1 + 1/c > 0$; in particular, $\Pr[X \geq x] \leq e^{-x}$ for $x \geq 7\lambda$,
2. for $\alpha \geq 0$,
   \[
   \Pr[X \leq (1 - \alpha)\lambda] \leq e^{-\alpha^2\lambda/2},
   \]
3. for $0 < \alpha \leq 3/2$,
   \[
   \Pr[|X - \lambda| \geq \alpha\lambda] \leq 2e^{-\alpha^2\lambda/3}.
   \]

For independent identically distributed random variable $X_1, \ldots, X_n$, where each $X_i$ is 1 with probability $p$, and 0 with probability $1 - p$, their sum $X = \sum_{i=1}^n X_i$ is distributed according to the *binomial distribution* with parameters $n$ and $p$. The expectation of $X$ is $\lambda = np$. The Chernoff bound [Che52] gives high concentration of $X$ around its expectation $\lambda$. In fact, all bounds of Theorem 6 hold also for the case of a binomially distributed random variable $X$ with expectation $\lambda$, and we will be using these bounds as the Chernoff bounds in the paper.

For pairwise independent identically distributed random variable $X_1, \ldots, X_n$, where each $X_i$ is 1 with probability $p$, and 0 with probability $1 - p$, their sum $X = \sum_{i=1}^n X_i$ is also concentrated around the expectation $\lambda = np$. The Chebyshev inequality bounds the deviation as follows: for $x > 0$, we have $\Pr[|X - \lambda| \geq x] \leq np(1 - p)/x^2$.

Finally, we will often use Markov-style averaging arguments. Two of the most common ones we need are as follows. For an event $E$ depending on independent random strings $x$ and $y$, if $\Pr_{x,y}[E] \geq \epsilon$, then for at least $\epsilon/2$ fraction of $xs$ it is the case that event $E$ holds for at least $\epsilon/2$ fraction of $ys$. Also, if $\Pr_{x,y}[E] \geq 1 - \gamma$, then for at least $1 - \sqrt{\gamma}$ fraction of $xs$ it is the case that event $E$ holds for at least $1 - \sqrt{\gamma}$ fraction of $ys$.

## 3 Overview and the proof of Main Theorem

Here we explain the general structure of our proof of Theorem 1. As mentioned in Introduction, we would like to use the learning algorithm *LEARN* of [IW97]. This algorithm can construct a circuit $C'$ approximately computing an $n$-variable Boolean function $f$ when given as input a circuit $C$ computing the direct product $f^k$ on a “non-trivial” fraction $\epsilon$ of inputs to $f^k$. This is a *learning* algorithm since it requires the values of the function $f$ on a few randomly independently chosen inputs $x_1, \ldots, x_t$, where $t$ is approximately $1/\epsilon^2$.

Getting these values is easy in the nonuniform setting of [IW97]. One just uses a simple counting argument to argue the existence of a small (polynomial in $n$ and $1/\epsilon$) set of inputs such that
algorithm \( \text{LEARN} \) works equally well when given the values of \( f \) on these fixed inputs. Then one can use nonuniform advice to specify these inputs and the corresponding values of \( f \).

In the uniform setting, we cannot use as advice the values of \( f \) on \( \text{poly}(n/\epsilon) \) inputs. Instead, we use the circuit \( C \) itself in order to generate sufficiently many random labeled examples of the form \( (x, f(x)) \) required by algorithm \( \text{LEARN} \). Then we can simply run algorithm \( \text{LEARN} \) on input \( C \) and the generated random examples, obtaining the requisite circuit \( C' \) that approximately computes \( f \). This approach is summarized in the diagram given in Figure 1 below.

**Figure 1. Schematic diagram of the decoding algorithm.**

![Diagram](image)

In reality, there are a number of technical issues one needs to handle in order to make the suggested approach work. The main issue is the extraction of random labeled examples \( (x, f(x)) \) from the circuit \( C \). A natural idea is to run \( C \) on a random \( k \)-tuple \( (x_1, \ldots, x_k) \), where each \( x_i \in \{0, 1 \}^n \), collect the \( k \)-tuple of outputs \( (b_1, \ldots, b_k) \) of \( C \), and finally output \( (x_1, b_1), \ldots, (x_k, b_k) \). Since \( C \) is assumed to compute \( f^k \) correctly on at least \( \epsilon \) fraction of input \( k \)-tuples, we get with probability \( \epsilon \) the output of the form \( (x_1, f(x_1)), \ldots, (x_k, f(x_k)) \).

Note that the given sampling algorithm produces a sequence of correct labeled examples only with probability \( \epsilon \). However, such a low probability is not a problem for us. If it were the case that with probability \( \text{poly}(\epsilon) \) our sampling algorithm produces enough random labeled examples, then we could conclude that with probability \( \text{poly}(\epsilon) \) the circuit \( C' \) produced by algorithm \( \text{LEARN} \) is good (i.e., it approximates \( f \) well). Indeed, conditioned on the event that our sampling algorithm produces random labeled examples, algorithm \( \text{LEARN} \) succeeds with high probability on input \( C \) and these examples; the correctness analysis of \( \text{LEARN} \) is given in [IW97]. Lifting the conditioning, we get that the circuit \( C' \) produced by \( \text{LEARN} \) is good with probability that is at least half of the probability that our sampling algorithm produces good examples, which is \( \text{poly}(\epsilon) \).

There are two problems with the suggested sampling algorithm. First, the \( k \) examples it produces (when successful) are correlated. They correspond to those \( \epsilon \) fraction of \( k \)-tuples of inputs where circuit \( C \) computes \( f^k \) correctly. The uniform distribution over these \( k \)-tuples is certainly not uniform over all \( k \)-tuples, unless \( \epsilon = 1 \). On the other hand, algorithm \( \text{LEARN} \) requires the values of \( f \) on \( t = \Omega(1/\epsilon^2) \) independent random inputs \( x_1, \ldots, x_t \).

The second problem is that, even if the samples produced by the suggested sampling algorithm were completely independent, there may not be enough of them. As mentioned above, for algorithm \( \text{LEARN} \) to work, we need to generate at least \( 1/\epsilon^2 \) labeled examples (actually, a bit more than
that, but we'll address this point later). If we are lucky, the given circuit $C$ $\epsilon$-computes $f^k$ for $k \geq 1/\epsilon^2$, and then the number of examples our sampling algorithm produces is enough. However, most likely $k < 1/\epsilon^2$, and so we need to do something else.

Next we explain our solution of these two problems.

### 3.1 Extracting almost independent random examples

Recall our problem. We are given a circuit $C$ that correctly computes the direct product $f^k$ on at least $\epsilon$ fraction of input $k$-tuples. We want to produce some number $t$ of independent random labeled examples $(x_1, f(x_1)), \ldots, (x_t, f(x_t))$.

Let $G \subseteq \{0,1\}^nk$ be the set of those $k$-tuples where $C$ correctly computes $f^k$. By our assumption, the weight of $G$ in the universe $\{0,1\}^nk$ is at least $\epsilon$. For each $i = 1, \ldots, k$, let $G_i$ be the projection of $G$ to the $i$th coordinate, i.e., $G_i$ is the set of $n$-bit strings that can occur in the $i$th position of a $k$-tuple from $G$. Clearly, it cannot be the case that all $G_i$s are simultaneously of weight less than $\epsilon^{1/k}$ (in the universe $\{0,1\}^n$), since in that case the weight of $G$ would be less than $\epsilon$.

To develop intuition, let us assume that $G$ is a direct product of $G_1, \ldots, G_k$ and that each $G_i$ is of weight at least $\epsilon^{1/k}$. Now consider the following modified sampling algorithm: Pick a random $k$-tuple $\bar{x} = (x_1, \ldots, x_k) \in \{0,1\}^nk$; run $C$ on input $\bar{x}$, obtaining the $k$-tuple of outputs $(b_1, \ldots, b_k)$; pick a random index $1 \leq j \leq k$; output the pair $(x_j, b_j)$.

Obviously, with probability $\epsilon$ the output produced by this new sampling algorithm is a correct example $(x, f(x))$. What is more, conditioned on the random $k$-tuple falling into the set $G$, the distribution of $x$ (i.e., the first element of the output pair $(x, f(x))$) is statistically close to uniform. The distance from the uniform distribution is at most $1 - \epsilon^{1/k}$, since we assumed that each $G_i$ is of weight at least $\epsilon^{1/k}$ and that $G$ is a direct product of $G_i$s. Observe that as $k$ gets larger, the distribution of $x$ gets closer to the uniform distribution.

Thus, the described sampling procedure allows us to produce (modulo some simplifying assumptions on the structure of $G$) a single random example $(x, f(x))$ with $x$ being distributed almost uniformly, conditioned on sampling a $k$-tuple from $G$.

To get more random examples, one might try running the described sampling procedure multiple times. However, the probability that $t$ independent runs of the sampling procedure produce $t$ good samples will be at most $\epsilon^t$ (since a single run succeeds with probability only about $\epsilon$). This is impractical unless $t$ is a constant, but in our case we need a super-constant $t > 1/\epsilon^2$.

A better way to sample more examples is as follows. View an input $k$-tuple as a $\sqrt{k}$-tuple of $\sqrt{k}$-tuples, i.e., $\bar{x} = (y_1, \ldots, y_{\sqrt{k}})$ where each $y_k$ is in $\{0,1\}^{n\sqrt{k}}$. Run a circuit $C$ on a random $\bar{x}$, and output a random $\sqrt{k}$-subtuple $y_j$ with the corresponding values of $C$ on $y_j$. The same analysis as above (again under the simplifying assumption on the structure of $G$) implies that this sampling procedure yields $\sqrt{k}$ examples $(x_1, f(x_1)), \ldots, (x_\sqrt{k}, f(x_\sqrt{k}))$ such that the entire tuple $x_1, \ldots, x_\sqrt{k}$ is distributed statistically close to uniform, conditioned on $\bar{x}$ falling in $G$. So, with probability $\epsilon$, we obtain $\sqrt{k}$ almost independent random labeled examples.

Our discussion above was assuming a nice structure of the set $G$. In general, $G$ is not so nicely structured, but, nonetheless, the given intuition is correct, and the sampling algorithm just described will still work, albeit with a slightly different bound on the statistical distance from the uniform distribution. We have the following Sampling Lemma, whose proof we postpone till later.

---

1 Under our simplifying assumptions on the structure of $G$, it is not necessary to pick $j$ at random. However, a random choice of $j$ will be useful for the case of an arbitrary $G$. 

---

9
**Lemma 7 (Sampling Lemma: Simple Case).** Let $C$ be a circuit that $\epsilon$-computes the direct product $f^k$, for some Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$. Let $\bar{x} = (x_1, \ldots, x_k) \in \{0,1\}^n$ be a uniformly random tuple, let $\bar{x}' = (x_{i_1}, \ldots, x_{i_k})$ be a uniformly random subtuple of $\bar{x}$, and let $b_{i_1} \cdots b_{i_k}$ be the values of $C(\bar{x})$ corresponding to the subtuple $\bar{x}'$. Then there is an event $E$ that occurs with probability at least $\epsilon$ such that, conditioned on $E$, the distribution of $\bar{x}'$ is $\alpha$-close to the uniform distribution over $\{0,1\}^{n\sqrt{k}}$, where $\alpha \leq 0.6\sqrt{\log(1/\epsilon)/\sqrt{k}}$.

### 3.2 Increasing the number of extracted random examples

Now we explain how to deal with problem that a given circuit $C$ $\epsilon$-computes the direct product $f^k$ for $k < 1/\epsilon^2$, and so $C$ is not immediately useful to obtain enough random examples required by algorithm $\text{LEARN}$. Let $t > 1/\epsilon^2$ be the actual number of random examples used by $\text{LEARN}$. Note that if it were the case that $k \geq t^2$, then we could use the sampler from Lemma 7 to obtain the requisite $t$ samples.

Suppose that $k < t^2$. In that case, what we would like to do is take the circuit $C$ that approximately computes $f^k$ and construct a new circuit $C^{\text{amp}}$ that approximately computes $f^{t^2}$. This can be viewed as amplifying the direct product computation: being able to approximately compute $f^k$ implies being able to approximately compute $f^{k'}$ for $k' > k$.

Unfortunately, we do not know how to achieve such direct product amplification. However, we can get something weaker, which is still sufficient for proving our Main Theorem. The circuit $C^{\text{amp}}$ we obtain will approximately compute a $(1 - \gamma^{\text{amp}})$-Direct Product $f^{k'}$, for some small parameter $\gamma^{\text{amp}}$. That is, on a certain fraction of input $k'$-tuples, the constructed circuit $C^{\text{amp}}$ will be correct in almost all positions $1 \leq i \leq k'$ rather than all positions. It turns out that we can also weaken our assumption on the initial circuit $C$ and also just require that $C$ approximately computes $(1 - \gamma)$-Direct Product $f^k$, for some $\gamma \geq 0$. We have the following lemma, whose proof we will give later in the paper.

**Lemma 8 (Direct Product Amplification).** For every $k \leq k' \in \mathbb{N}$, $\gamma > 0$, and $\epsilon > e^{-k^{0.001}}$, there is a randomized algorithm $A$ with the following property. Let $f$ be an $n$-variable Boolean function such that a circuit $C$ $\epsilon$-computes $(1 - \gamma)$-Direct Product $f^k$. Given $C$, algorithm $A$ outputs with probability at least $\epsilon'$ a circuit $C^{\text{amp}}$ that $\epsilon'$-computes $(1 - \gamma')$-Direct Product $f^{k'}$, where

- $\epsilon' = \epsilon^{\text{poly log}_k k'}$, and
- $\gamma' \leq (\gamma + k^{-0.4})\text{poly log}_k k'$.

The running time of $A$, and hence also the size $|C^{\text{amp}}|$ of $C^{\text{amp}}$, is at most $(|C|/\epsilon)^{\text{poly log}_k k'}$.

Observe that for the case where $\epsilon > \text{poly}(1/k)$, the number of random examples we need to run algorithm $\text{LEARN}$ is about $1/\epsilon^2 = \text{poly}(k)$. So we need to amplify direct product for $f$ from $k$ to $k' = \text{poly}(k)$. For such $\epsilon$ and $k'$, the Direct Product Amplification theorem above yields that $\epsilon' = \text{poly}(\epsilon)$, the running time of $A$ is at most $\text{poly}(|C|/\epsilon)$, and $\gamma' \leq O(\gamma + k^{-0.4})$. Assuming that $\gamma < k^{-0.4}$, we get that $\gamma' < k^{-0.3}$.

The only remaining question is how to use this circuit $C^{\text{amp}}$ to extract random examples. Our sampling algorithm from Lemma 7 requires access to a circuit for a direct product of $f$ that is correct on all positions for at least $\epsilon$ fraction of input tuples, rather than just almost all positions. What kind of samples do we get when using such an imperfect direct-product circuit $C^{\text{amp}}$? Are these samples going to be good enough to run algorithm $\text{LEARN}$? It turns out that the answer to the last question is yes, once we modify algorithm $\text{LEARN}$ appropriately. We shall give more details in the next subsection.
3.3 Algorithm LEARN with faulty examples

It is not difficult to argue (and we will formalize it later) that the same sampling algorithm of Lemma 7 will produce a sequence of samples \((x_1, b_1), \ldots, (x_t, b_t, \bar{x})\) such that, conditioned on a certain event \(E\) that occurs with probability \(\Omega(\epsilon)\), the distribution of all \(x_i\)s is statistically close to uniform, and for almost all \(1 \leq i \leq \sqrt{k}\) it is the case that \(b_i = f(x_i)\). That is, almost all of the produced pairs \((x_i, b_i)\) will be correct labeled examples \((x_i, f(x_i))\). Can one use these slightly imperfect examples as input to algorithm LEARN? We show that one indeed can, after appropriately modifying the algorithm LEARN to take into account potential inaccuracy in provided random examples.

Before stating the result about our modification of algorithm LEARN from [IW97], we outline how the original algorithm from [IW97] works. Given a circuit \(C\) that \(\epsilon\)-computes \(f^k\), the circuit \(C'\) approximately computing \(f\) does the following: On input \(z\), randomly sample \(t(k-1)\) labeled examples \((x, f(x))\), where \(t\) is a parameter of the algorithm. Think of these examples as \(t\) blocks of \((k-1)\) pairs \((x, f(x))\) each, so that the \(i\)th block is \((x_i, f(x_i)), \ldots, (x_{i,k-1}, f(x_{i,k-1}))\). For every block \(1 \leq i \leq t\), pick a random position \(1 \leq j_i \leq k\), and form a \(k\)-tuple \(x_i = (x_{i,1}, \ldots, x_{i,j_i-1}, z, x_{i,j_i}, \ldots, x_{i,k-1})\), with \(z\) in the \(j_i\)th position. Run circuit \(C\) on the tuple \(\bar{x}_i\). Depending on the number of correct answers \(C\) gives in positions other than \(j_i\), probabilistically assign the variable \(v_i\) either the answer of \(C\) in position \(j_i\), or a random bit. Finally, output the majority value over all \(v_i\), for \(1 \leq i \leq t\).

The key parameter of this algorithm is the probability with which to believe or not to believe the answer of \(C\) in the \(j_i\)th position of block \(i\). That probability is proportional to the number of correct answers \(C\) gives for the other positions in the block; note that we can verify the correctness of \(C\)'s answers in these other positions since we know the corresponding values of \(f\).

Suppose we use the same algorithm as above to construct \(C'\), but with imperfect samples such that, for each block \(i\), we know the values of \(f\) on almost all strings \(x_{i,1}, \ldots, x_{i,k-1}\) in the block. It is possible to modify the value of the probability with which one believes the answer of \(C\) in the \(j_i\)th position so that the output circuit \(C'\) still approximates the function \(f\) well.

Before stating the actual result, we need some definitions. For parameters \(k, n, t\), let \(D\) be a probability distribution over \(t\) blocks where each block consists of \(k-1\) pairs \((x, b_x)\), with \(x \in \{0, 1\}^n\) and \(b_x \in \{0, 1\}^t\); that is, \(D\) is a probability distribution over \((\{0, 1\}^n \times \{0, 1\})^{(k-1)t}\). Think of \(D\) as a distribution over \(t\) blocks of \((k-1)\) samples required by algorithm LEARN. Let \(D_x\) be the probability distribution over \((\{0, 1\}^n)^{(k-1)t}\) obtained from \(D\) by keeping only \(x\) from every pair \((x, b_x)\); that is, \(D_x\) is the distribution over the inputs to \(f\) in all \(t\) blocks. For parameters \(0 \leq \kappa, \gamma \leq 1\), we say that the distribution \(D\) is \((\gamma, \kappa)\)-good if

1. for every sample from \(D\), each block \((x_1, b_1), \ldots, (x_{k-1}, b_{k-1})\) of the sample is such that \(b_j = f(x_j)\) for at least \((1 - \gamma)\) fraction of \(j \in [k-1]\), and
2. the distribution \(D_x\) is statistically \(\kappa\)-close to the uniform distribution over \((\{0, 1\}^n)^{(k-1)t}\).

The next lemma says that an appropriate modification of the algorithm from [IW97] works when given samples from a distribution \(D\) that is good according to the definition above.

**Lemma 9 (Analysis of Algorithm LEARN).** For any \(\mu \geq \nu > 0\), \(\kappa > 0\), \(\epsilon > e^{-k^{\nu/3}}\), \(\rho = \frac{\log(1/\kappa)}{\log(1/\epsilon)} + k^{-\nu/2}\), and \(t = O((\log 1/\rho)/\epsilon^2)\), there is a probabilistic algorithm LEARN satisfying the following. Let \(f\) be an \(n\)-variable Boolean function such that a circuit \(C\) \(\epsilon\)-computes \((1-k^{-\mu})\)-Direct Product \(f^k\), and let \(D\) be a probability distribution over \((\{0, 1\}^n \times \{0, 1\})^{(k-1)t}\) that is \((k^{-\nu}, \kappa)\)-good. Then algorithm LEARN, given as input \(C\) and a random sample from \(D\), outputs with probability
at least $1 - O(\rho) - \kappa$ a circuit $C'$ that computes $f$ on at least $1 - O(\rho)$ fraction of inputs. The running time of LEARN, and hence also the size of the circuit $C'$, is at most $\text{poly}(|C|, 1/\epsilon)$.

3.4 Proof of Main Theorem

Now we can give the proof of our Main Theorem (Theorem 1), using the lemmas above; the proofs of these lemmas will be given in later sections of the paper.

The plan is to use Direct Product Amplification of a given circuit $C$ to obtain a new circuit $C^{\text{amp}}$ for a larger direct product of $f$, extract random labeled examples from $C^{\text{amp}}$, and finally run LEARN on $C$ and extracted random examples. One technicality is that we need to extract examples from a circuit $C^{\text{amp}}$ that computes only $(1 - \gamma)$-Direct Product of $f$, for some $\gamma > 0$, whereas our Sampling Lemma (Lemma 7) is for the case where $\gamma = 0$. So we first generalize our Sampling Lemma as follows.

**Lemma 10 (Sampling Lemma: General Case).** Let $C$ be a circuit that $\epsilon$-computes $(1 - \gamma)$-direct product $f^k$, for some Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Let $\bar{x} = (x_1, \ldots, x_k) \in \{0, 1\}^{nk}$ be a uniformly random tuple, let $\bar{x}' = (x_{i_1}, \ldots, x_{i_{\gamma k}})$ be a uniformly random subtuple of $\bar{x}$, and let $b_{i_1} \ldots b_{i_{\gamma k}}$ be the values of $C(\bar{x})$ corresponding to the subtuple $\bar{x}'$. Then there is an event $E$ that occurs with probability $\Omega(\epsilon)$ so that, conditioned on $E$, the following two conditions hold:

1. the distribution of $\bar{x}'$ is statistically $\alpha$-close to the uniform distribution over $\{0, 1\}^{nk}$, where $\alpha \leq 0.6 \sqrt{\log(1/\epsilon)/\sqrt{k} + e^{-\Omega(k^{0.1})}}$, and

2. for all but at most $O(\gamma) + k^{-0.4}$ fraction of elements $x_{i_j}s$ in $\bar{x}'$, it is the case that $b_{i_j} = f(x_{i_j})$.

Now we can continue with our proof of Theorem 1. Let $C$ be a given circuit that $\epsilon$-computes the $(1 - k^{-\mu})$-Direct Product $f^k$ of an $n$-variable Boolean function $f$, where $\mu > 0$ and $\epsilon > e^{-k^\alpha}$ for $\alpha = \min\{\mu/1000, 0.0001\}$. Set $\nu = 0.9 \times \min\{\mu, 0.4\}$. Let $\rho$ and $t$ be as in the statement of Lemma 9; i.e., $t(k - 1)$ is the total number of labeled examples required by the modified algorithm LEARN of Lemma 9. Set $k' = (t(k - 1))^2$. We do the following.

1. Apply the Direct Product Amplification algorithm of Lemma 8 to the circuit $C$, with the parameters $k, k'$ as above and $\gamma = k^{-\mu}$. We get with probability $\epsilon' = e^{\text{poly} \log_k k'}$ a new circuit $C^{\text{amp}}$ that $\epsilon'$-computes $(1 - \gamma^{\text{amp}})$-Direct Product $f^{k'}$, for $\gamma^{\text{amp}} \leq (k^{-\mu} + k^{-0.4})\text{poly} \log_k k'$. The quantity $\text{poly} \log_k k'$ can be upperbounded by $k^{\min\{\mu/100, 0.001\}}$. So, $\gamma^{\text{amp}} \leq k^{-0.99 \min\{\mu, 0.4\}}$.

2. Apply the Sampling Lemma (Lemma 10) to the circuit $C^{\text{amp}}$, obtaining $\sqrt{k'} = t(k - 1)$ samples of the form $(x, b_x) \in \{0, 1\}^n \times \{0, 1\}$. With probability $\Omega(\epsilon')$ an event $E$ occurs such that, conditioned on $E$, the distribution on $xs$ is statistically $0.6 \sqrt{\log(1/\epsilon')/\sqrt{k'} = o(1)}$-close to uniform, and all but at most $\gamma' = O(\gamma^{\text{amp}}) + k^{-0.4}$ fraction of samples are correct labeled examples of the form $(x, f(x))$.

3. Randomly partition the $t(k - 1)$ samples into $t$ blocks. This will ensure that each block has at most $2\gamma'$ incorrect examples, with high probability. Indeed, for a fixed block, the probability that it gets more than twice the expected number $\gamma'(k - 1)$ of incorrect examples is, by the Hoeffding bound of Theorem 6, at most $e^{-\Omega(\gamma'(k - 1))}$. The latter is at most $e^{-\Omega(k^{0.6})}$ since $\gamma' \geq k^{-0.4}$. By the union bound, the probability that at least one of the $t$ blocks gets more than twice the expected number of incorrect examples is at most $te^{-\Omega(k^{0.6})}$.
For \( t = O((\log 1/\rho)/\varepsilon^2) \), \( \rho = (\log 1/\varepsilon)/k + k^{-\nu/2} \), and \( \varepsilon \geq e^{-k^{0.001}} \), we get \( \rho \leq k^{-\Omega(1)} \) and \( t \leq O(\log k/\varepsilon^2) \). Hence, the probability that any block gets more than \( 2\gamma' \leq k^{-\nu} \) fraction of incorrect examples is at most \( O(\log k/\varepsilon^2) \cdot e^{-\Omega(k^{0.6})} \), which is exponentially small in \( k^{\Omega(1)} \).

4. Suppose that event \( E \) occurred. Then the last two steps ensure that we have with probability \( 1 - e^{-k^{\Omega(1)}} \) a sample from a \((k^{-\nu}, o(1))\)-good distribution over \( t \) blocks of \((k - 1)\) pairs \((x, b_x)\). Applying algorithm LEARN from Lemma 9 to the circuit \( C \) and this sample, we get with probability \( 1 - O(\rho) - o(1) = 1 - o(1) \) a circuit \( C' \) that \((1 - O(\rho))\)-computes \( f \). Lifting the conditioning on \( E \), we conclude that such a circuit \( C' \) is produced with probability at least \( \Omega(\varepsilon') \).

The running time of the algorithm described in the four steps above is at most \((|C|/\varepsilon)^{\text{poly} \log k} \), which can be upperbounded by \((|C|/\varepsilon)^{\text{poly} \log k} 1/\varepsilon \). The probability \( \Omega(\varepsilon') \) of getting a good circuit \( C' \) can be lowerbounded by \( \varepsilon^{\text{poly} \log k} 1/\varepsilon \).

This finishes the proof of Theorem 1, modulo the proofs of Lemmas 8, 9, and 10, which will be given in the following sections of the paper.

4 Our tools

4.1 Proof of the Sampling Lemma

Here we give the proof of Lemma 10. For the ease of reference, we re-state this lemma below.

**Lemma 10 (Sampling Lemma: General Case).** Let \( C \) be a circuit that \( \varepsilon \)-computes \((1 - \gamma)\)-direct product \( f^k \), for some Boolean function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \). Let \( \bar{x} = (x_1, \ldots, x_k) \in \{0, 1\}^{nk} \) be a uniformly random tuple, let \( \bar{x}' = (x_1, \ldots, x_{\sqrt{k}}) \) be a uniformly random subtuple of \( \bar{x} \), and let \( b_{i_1} \ldots b_{i_{\sqrt{k}}} \) be the values of \( C(\bar{x}) \) corresponding to the subtuple \( \bar{x}' \). Then there is an event \( E \) that occurs with probability \( \Omega(\varepsilon) \) so that, conditioned on \( E \), the following two conditions hold:

1. the distribution of \( \bar{x}' \) is statistically \( \alpha \)-close to the uniform distribution over \( \{0, 1\}^{n\sqrt{k}} \), where \( \alpha \leq 0.6 \sqrt{\log(1/\varepsilon)/\sqrt{k} + e^{-\Omega(k^{0.1})}} \), and
2. for all but at most \( O(\gamma) + k^{-0.4} \) fraction of elements \( x_{ij} \) in \( \bar{x}' \), it is the case that \( b_{ij} = f(x_{ij}) \).

We need the following result implicit in [Raz98].

**Lemma 11.** Let \( G \subseteq \{0, 1\}^{mn} \) be any subset of size \( \epsilon 2^{mn} \). Let \( U \) be a uniform distribution on the set \( \{0, 1\}^n \), and let \( D \) be the distribution defined as follows: pick a tuple \((x_1, \ldots, x_m)\) of \( n \)-bit strings uniformly from the set \( G \), pick an index \( i \) uniformly from \([m]\), and output \( x_i \). Then \( \text{Dist}(U, D) \leq 0.6 \sqrt{\log 1/\varepsilon \over m} \).

**Proof.** Let \( \bar{X} = (X_1, \ldots, X_m) \) be random variables drawn according to the uniform distribution over the set \( G \). Let \( r \in [m] \) be a uniformly distributed random variable. Consider the random variable \( X = X_r \). We will argue that the distribution of \( X \) is statistically close to the uniform distribution \( U \).

First observe that \( X = X_r \) is distributed according to the average of the distributions of \( X_1, \ldots, X_m \), i.e., \( \Pr[X = x] = \frac{1}{m} \sum_{i=1}^m \Pr[X_i = x] \). By the concavity of the entropy function, we obtain
\[
H(X) \geq \frac{1}{m} \sum_{i=1}^m H(X_i).
\]
By the independence bound on entropy, the sum of entropies is lowerbounded by the joint entropy, and so we get
\[ H(X) \geq \frac{1}{m} H(X_1, \ldots, X_m). \] (1)
Since \( \bar{X} = (X_1, \ldots, X_m) \) is uniformly distributed over \( G \), its Shannon entropy is exactly \( \log_2 |G| = mn - \log_2(1/\epsilon) \). Combining this with Equation (1) above, we get
\[ H(X) \geq n - \frac{\log_2 1/\epsilon}{m}. \]
Finally, we use Lemma 5 above to conclude that
\[ \|X - U\|_1 \leq \sqrt{(2 \ln 2) \log_2 1/\epsilon / \sqrt{k}}. \]
Since the statistical distance between \( X \) and \( U \) is half of the \( \ell_1 \) norm above, we get the claimed bound.

Now we can prove Lemma 10.

**Proof of Lemma 10.** Think of each \( k \)-tuple \( \bar{x} \) as a \( \sqrt{k} \)-tuple whose elements are \( \sqrt{k} \)-tuples (any fixed partitioning suffices). Let \( G \) be the set of those \( \epsilon \)-fraction of \( k \)-tuples \( \bar{x} \) where \( C \) computes \( f^k \) for all but \( \gamma \) fraction of positions. A random \( k \)-tuple \( \bar{x} \) falls into the set \( G \) with probability at least \( \epsilon \). Call this event \( E_1 \). Conditioned on \( E_1 \), this tuple is uniformly distributed over \( G \) and hence, by Lemma 11, we conclude that a random element of \( \bar{x} \) (i.e., a random \( \sqrt{k} \)-subtuple of \( \bar{x} \)) is distributed almost uniformly, with the statistical distance from the uniform distribution less than
\[ 0.6 \sqrt{\log(1/\epsilon) / \sqrt{k}}. \]
Observe that every tuple in \( G \) has at most \( \gamma \) fraction of “bad” elements where \( C \) disagrees with the function \( f \). If we pick a random subtuple, then the probability that it contains more than \( 7 \gamma + k^{-0.4} \) fraction of bad elements will be at most \( e^{-(7 \gamma + k^{-0.4}) \sqrt{k}} \leq e^{-k^{0.1}} \) by the Hoeffding bound of Theorem 6.

Let \( E_2 \) denote the event that a random subtuple contains at most \( 7 \gamma + k^{-0.4} \) fraction of “bad” elements. We get that conditioned on \( E_1 \), the probability of \( E_2 \) is at least \( 1 - e^{-k^{0.1}} \). Let the event \( E \) be the conjunction of \( E_1 \) and \( E_2 \). Clearly, the probability of \( E \) is at least \( \epsilon(1 - e^{-k^{0.1}}) = \Omega(\epsilon) \). It is also clear that conditioned on \( E \), the output subtuple satisfies item (2) of the lemma. Finally, it is easy to argue that since \( E_2 \) has high probability conditioned on \( E_1 \), the distribution of random elements of \( \bar{x} \) conditioned on \( E \) has statistical distance from the uniform distribution at most
\[ 0.6 \sqrt{\log(1/\epsilon) / \sqrt{k}} + e^{-\Omega(k^{0.1})}. \]
Hence, item (1) of the lemma also holds. \( \square \)

### 4.2 Proof of the Direct Product Amplification

Here we will prove Lemma 8. For convenience, we re-state it below.

**Lemma 8 (Direct Product Amplification).** For every \( k \leq k' \in \mathbb{N}, \gamma > 0 \), and \( \epsilon > e^{-k^{0.001}} \), there is a randomized algorithm \( A \) with the following property. Let \( f \) be any \( n \)-variable Boolean function such that a circuit \( C \) \( \epsilon \)-computes \( (1 - \gamma) \)-Direct Product \( f^k \). Given \( C \), algorithm \( A \) outputs with probability at least \( \epsilon' \) a circuit \( C^{\text{amp}} \) that \( \epsilon' \)-computes \( (1 - \gamma') \)-Direct Product \( f^{k'} \), where
- \( \epsilon' = e^{\text{poly} \log_k k'} \), and
- \( \gamma' \leq (\gamma + k^{-0.4}) \text{poly} \log_k k' \).
The running time of $A$, and hence also the size $|C^{\text{amp}}|$ of $C^{\text{amp}}$, is at most $(|C|/\epsilon)^{\text{polylog} k'}$. 

Our proof of Lemma 8 will be recursive. We first show how to get from a circuit approximately computing the direct product $f^k$ a new circuit that approximately computes $f^{k^{1.5}}$. Then we iterate this one-step amplification $O(\log \log k')$ times, obtaining a circuit for $f^{k'}$.

We state the lemma about one-step amplification next.

**Lemma 12 (One-step Direct Product Amplification).** There is a randomized algorithm $A$ with the following property. Let $f$ be any $n$-variable Boolean function such that a circuit $C$ $\varepsilon$-computes $(1 - \gamma)$-Direct Product $f^k$, where $\varepsilon > e^{-k^{0.001}}$. Given $C$, algorithm $A$ outputs with probability at least $\varepsilon'$ a circuit $C'$ $\varepsilon'$-computing $(1 - \gamma')$-Direct Product $f^{k'}$, where $\varepsilon' = \Omega(\varepsilon^2)$, $\gamma' \leq O(\gamma + k^{-0.4})$, and $k' = k^{3/2}$. The running time of $A$, and hence also the size of $C'$, is polynomial in the size of $C$ and $1/\varepsilon$.

Applying Lemma 12 for $\log_{1.5} \log k'$ times, one easily obtains Lemma 8. The rest of this section is devoted to the proof of Lemma 12.

Let $G \subseteq \{0, 1\}^{nk}$ be the set of those $k$-tuples where the circuit $C$ correctly computes $f^k$ in all but at most $\gamma$ fraction of positions $1 \leq j \leq k$. In the proof, we will focus on certain $k' = k^{1.5}$-tuples $\bar{x}$ that have the property that a significant fraction of their subtuples of size $k$ fall in the set $G$.

We need the following definitions. For a $k'$-tuple $\bar{x}$ and a subset $S \subseteq \{1, \ldots, k'\}$, we denote by $\bar{x}|_S$ the restriction of $\bar{x}$ to the positions in $S$; so $\bar{x}|_S$ is a tuple of size $|S|$. For a given $k'$-tuple $\bar{x}$, we say that a size-$k$ set $S \subseteq \{1, \ldots, k'\}$ is $\alpha$-good if $C(\bar{x}|_S)$ disagrees with $f^k(\bar{x}|_S)$ in at most $\alpha$ fraction of positions.

The following claim says that there are many $k'$-tuples that have many $\gamma$-good sets.

**Claim 13.** There are at least $\varepsilon/2$ fraction of $k'$-tuples $\bar{x}$ such that

$$\Pr_{S \subseteq \{1, \ldots, k'\} : |S| = k}[S \text{ is } \gamma\text{-good}] \geq \varepsilon/2,$$

where the probability is over uniformly random size-$k$ subsets $S$ of the set $\{1, \ldots, k'\}$.

**Proof.** Note that for a random $k'$-tuple $\bar{x}$ and a random size-$k$ subset $S$, the tuple $\bar{x}|_S$ is uniformly distributed in $\{0, 1\}^{nk}$, and so it falls into $G$ with probability at least $\varepsilon$. A simple averaging argument completes the proof. \hfill \Box

For the rest of the proof, we fix one particular $k'$-tuple $\bar{x}$ satisfying Eq. (2) of Claim 13.

We now give an informal description of our algorithm in Lemma 12. First, we randomly choose a $k$-size set $S$, run the circuit $C$ on $\bar{x}|_S$, and use the obtained values as if they were equal to $f^k(\bar{x}|_S)$. This roughly gives us the values of $f$ on a portion of $\bar{x}$. To obtain the values of $f$ in the positions of $\bar{x}$ outside the set $S$, we repeatedly do the following. Pick a random $k$-size set $T$ and run $C$ on $\bar{x}|_T$. If the values $C(\bar{x}|_S)$ and $C(\bar{x}|_T)$ are mostly consistent on inputs $\bar{x}|_{S \cap T}$, then we trust that the values $C(\bar{x}|_T)$ are mostly correct, and take one such value for a random position in $T$. This gives us a vote for the value of $f$ on another input in $\bar{x}$. Repeating this process enough times, we collect many votes for many positions in $\bar{x}$. For each position of $\bar{x}$ where we have enough votes, we take the majority value as our guess for the value $f$ in that position. For those (few) positions where we don't have enough votes, we assign the value of $f$ arbitrarily, say by flipping a coin.

The formal description of the algorithm is given in Figure 1.

For the analysis of the algorithm, we will need the following definition that formalizes what we mean when we say that for two sets $S$ and $T$ the values $C(\bar{x}|_S)$ and $C(\bar{x}|_T)$ are mostly consistent in positions $S \cap T$. For two size-$k$ subsets $S = \{j_1, \ldots, j_k\}$ and $T = \{j'_1, \ldots, j'_k\}$ of $[k']$, let
1. For every $i \in [k']$, set $Answers_i = \text{empty string}$.
2. Choose $S \subseteq [k']$ of size $|S| = k$ uniformly at random.
3. Let $S = \{j_1, \ldots, j_k\}$ where $1 \leq j_1 < j_2 < \cdots < j_k \leq k'$.
4. Run $C(\bar{x}|S) = (b_{j_1} \ldots b_{j_k})$, where $\bar{x}|S$ denotes the restriction of $\bar{x}$ to the coordinates from $S$.
5. Repeat lines 6–11 for at most $\text{timeout}$ times:
   6. Choose $T \subseteq [k']$ of size $|T| = k$ uniformly at random.
   7. Let $T = \{j'_1, \ldots, j'_k\}$ where $1 \leq j'_1 < j'_2 < \cdots < j'_k \leq k'$.
   8. Compute $C(\bar{x}|T) = (t_{j'_1} \ldots t_{j'_k})$.
   9. Let $m = |\{j \in S \cap T \mid b_j \neq t_j\}|/|S \cap T|$.
   10. If $m < \rho$, $\%$ if $T$ is accepted w.r.t. $S$
   11. then choose a random $i \in T$ and extend the string $Answers_i$ with the bit $t_i$.
12. For every $i \in [k']$, let $\text{count}_i = |Answers_i|$.
13. Let $\text{total} = \sum_{i \in [k']} \text{count}_i$.
14. If $\text{count}_i < \frac{\text{total}}{2k'}$, then set $output_i = r_i$ for a random bit $r_i$;
15. else set $output_i = \text{Majority} Answers_i$.

**Algorithm 1:** Direct Product amplifier

$C(\bar{x}|S) = (b_{j_1} \ldots b_{j_k})$ and let $C(\bar{x}|T) = (t_{j'_1} \ldots t_{j'_k})$. We say that a set $T$ is accepted w.r.t. $S$ if $|\{j \in S \cap T \mid b_j \neq t_j\}|/|S \cap T| < 16\gamma + 3k^{-0.4}$.

We will argue that for most $\gamma$-good sets $S$,

1. **[Completeness]** almost every $\gamma$-good set $T$ is accepted with respect to $S$, and
2. **[Soundness]** almost every set $T$ accepted with respect to $S$ is $O(\gamma + k^{-0.4})$-good.

**Claim 14 (Completeness).** For at least $1 - e^{-\Omega(k^{0.1})}$ fraction of $\gamma$-good sets $S$,

$$\Pr_{\gamma \text{-good } T}[T \text{ is accepted w.r.t. } S] \geq 1 - e^{-\Omega(k^{0.1})}. $$

**Proof.** First we show that, conditioned on both $T$ and $S$ being $\gamma$-good random sets,

$$\Pr_{S,T}[T \text{ is accepted w.r.t. } S] \geq 1 - e^{-\Omega(k^{0.1})}. $$  \hspace{1cm} (3)

The conclusion of the claim will then follow from Eq. (3) by a straightforward averaging argument.

For $S$, let us denote by $S_B$ the subset of all those elements $j \in S$ where $C(\bar{x}|S)_j$ is wrong. Define $S_G = S \setminus S_B$. Similarly, let $T_B$ denote the subset of $T$ where the circuit $C$ makes mistakes, and let $T_G = T \setminus T_B$. By our assumption that $S$ and $T$ are $\gamma$-good, we have that both $S_B$ and $T_B$ have weight at most $\gamma$ in their respective sets. Note that the errors in $S \cap T$ can only come from $S_G \cap T_B$ and $T_G \cap S_B$. We’ll upperbound the sizes of these sets by upperbounding the sizes of bigger sets $S \cap T_B$ and $T \cap S_B$, respectively.

Fix a $\gamma$-good set $T$. Note that $T_B$ has density at most $\gamma k/k' = \gamma / \sqrt{k}$ in the universe $\{1, \ldots, k'\}$. A random (not necessarily $\gamma$-good) size-$k$ subset $S$ of $\{1, \ldots, k'\}$ is expected to intersect $T_B$ in at most $k\gamma / \sqrt{k} = \gamma \sqrt{k}$ places. By the Hoeffding bound of Theorem 6, we get that

$$\Pr_S[|S \cap T_B| > (7\gamma + k^{-0.4})\sqrt{k}] \leq e^{-k^{0.1}}.$$
Conditioned on $S$ being a $\gamma$-good set, this probability will be at most factor $2/\epsilon$ larger (since $\gamma$-good sets have weight at least $\epsilon/2$ in $[k']$). Since $\epsilon > e^{-k^{0.001}}$, the resulting conditional probability is still at most $e^{-\Omega(k^{0.1})}$.

A symmetric argument shows that for all but an exponentially small fraction of $\gamma$-good sets $T$, the intersection $T \cap S_B$ will be at most $(7\gamma + k^{-0.4})\sqrt{k}$ for any fixed $\gamma$-good set $S$. Thus, overall, for all but an exponentially small fraction of $\gamma$-good sets $S$ and $T$, the set $(S \cap T) \cup (T \cap S_B)$ will be of size at most $2(7\gamma + k^{-0.4})\sqrt{k}$.

On the other hand, applying the Hoeffding bound of Theorem 6 to the size of the set $S \cap T$, we get that for all but an exponentially small fraction of $\gamma$-good $S$ and $T$, $(1 - 0.1)\sqrt{k} \leq |S \cap T| \leq (1 + 0.1)\sqrt{k}$. Thus, with probability at least $1 - e^{-\Omega(k^{0.1})}$ over $\gamma$-good sets $S$ and $T$, the fraction of indices in $S \cap T$ where the circuit $C$ gives inconsistent answers will be less than $16\gamma + 3k^{-0.4}$, and so $T$ will be accepted w.r.t. $S$.

**Claim 15 (Soundness).** For at least $1 - e^{-\Omega(k^{0.1})}$ fraction of $\gamma$-good sets $S$, at least $1 - e^{-\Omega(k^{0.1})}$ fraction of sets $T$ that are accepted w.r.t. $S$ are $\alpha = (29\gamma + 6k^{-0.4})$-good.

**Proof.** As in the proof of Claim 14 above, we'll denote by $S_B$ and $S_G$ the subsets of indices of $S$ where the circuit $C$ makes mistakes and is correct, respectively. Similarly we denote by $T_B$ and $T_G$ the corresponding subsets of $T$. We will show that

$$\Pr_{T, \gamma\text{-good } S} [|T_B| > \alpha k \mid T \text{ is accepted w.r.t. } S] \leq e^{-\Omega(k^{0.1})}. \tag{4}$$

The claim will then follow by a simple averaging argument.

The conditional probability in Eq. (4) can be equivalently written as

$$\frac{\Pr_{T, \gamma\text{-good } S} [|T_B| > \alpha k \text{ and } T \text{ is accepted w.r.t. } S]}{\Pr_{T, \gamma\text{-good } S} [T \text{ is accepted w.r.t. } S]} \tag{5}$$

Since by our assumption $\gamma$-good sets $T$ have weight at least $\epsilon/2$ in $[k']$, we can lowerbound the probability in the denominator of (5) by

$$(\epsilon/2)\Pr_{\gamma\text{-good } T, \gamma\text{-good } S} [T \text{ is accepted w.r.t. } S].$$

By Eq. (3), this is at least $(\epsilon/2)(1 - e^{-\Omega(k^{0.1})}) \geq \epsilon/3$. It remains to upperbound the probability in the numerator of (5).

Recall that $T$ is accepted w.r.t. $S$ if both $|T_B \cap S_G|$ and $|T_G \cap S_B|$ are small relative to $|S \cap T|$. We will consider only $T_B \cap S_G$, and show that with high probability either $|T_B| \leq \alpha k$ or $T_B \cap S_G$ is large relative to $|S \cap T|$. Since the latter implies that $T$ is not accepted w.r.t. $S$, we will be done with upperbounding the numerator of (5).

First we bound $|S \cap T|$. For every fixed $\gamma$-good set $S$, a random set $T$ is expected to intersect $S$ in $\frac{k}{2} \approx \sqrt{k}$ places. By the Hoeffding bound of Theorem 6, $\Pr_T [|T \cap S| \geq 1.1\sqrt{k}] \leq e^{-\Omega(\sqrt{k})}$. Since this is true for every $\gamma$-good set $S$, it is also true for a random $\gamma$-good set $S$. That is,

$$\Pr_{T, \gamma\text{-good } S} [|T \cap S| \geq 1.1\sqrt{k}] \leq e^{-\Omega(\sqrt{k})}. \tag{6}$$

Next we observe that $|T_B \cap S_G| = |T_B \cap S| - |T_B \cap S_B| \geq |T_B \cap S| - |T \cap S_B|$. Using the Hoeffding bound, we will upperbound $|T \cap S_B|$ and lowerbound $|T_B \cap S|$. For every fixed $\gamma$-good set $S$, a random size-$k$ set $T$ is expected to intersect $S_B$ in at most $\frac{\sqrt{k}}{k} k = \gamma\sqrt{k}$ positions. By the Hoeffding bound of Theorem 6,

$$\Pr_T [|T \cap S_B| > (7\gamma + k^{-0.4})\sqrt{k}] \leq e^{-k^{0.1}}. \tag{7}$$
Since Eq. (7) holds for every $\gamma$-good $S$, it also holds for a random $\gamma$-good $S$. That is, we have

$$\Pr_{T,\gamma}\text{-good } S \left| |T \cap S_B| > (7\gamma + k^{-0.4})\sqrt{k} \right| \leq e^{-k^{0.1}}. \tag{8}$$

For every fixed set $T_B$ such that $|T_B| > \alpha k$, a random size-$k$ set $S$ is expected to intersect $T_B$ in more than $\frac{\alpha k}{k} = \alpha \sqrt{k}$ places. By the Hoeffding bound of Theorem 6,

$$\Pr_S[|S \cap T_B| \leq 0.9\alpha \sqrt{k}] \leq e^{-\Omega(\alpha \sqrt{k})} \leq e^{-\Omega(k^{0.1})},$$

where the latter inequality is because $\alpha \geq k^{-0.4}$. Since $\gamma$-good sets have weight at least $\epsilon/2$ in the universe of all size-$k$ sets and since $\epsilon > e^{-k^{0.001}}$, we get

$$\Pr_{\gamma}\text{-good } S \left| |S \cap T_B| \leq 0.9\alpha \sqrt{k} \right| \leq (2/\epsilon)e^{-\Omega(k^{0.1})} \leq e^{-\Omega(k^{0.1})}. \tag{9}$$

It is easy to see that inequality (9) implies

$$\Pr_{T,\gamma}\text{-good } S \left| |S \cap T_B| < 0.9\alpha \sqrt{k} \text{ and } |T_B| > \alpha k \right| \leq e^{-\Omega(k^{0.1})}. \tag{10}$$

Combining Eqs. (6), (8), and (10), we get that with probability at least $1 - e^{-\Omega(k^{0.1})}$,

1. $|T_B| \leq \alpha k$, or
2. $|S \cap T| < 1.1\sqrt{k}$, $|T \cap S_B| \leq (7\gamma + k^{-0.4})\sqrt{k}$, and $|T_B \cap S| \geq 0.9\alpha \sqrt{k}$.

In the second case, we have $(|T_B \cap S| - |T \cap S_B|)/|S \cap T| \geq (0.9\alpha - 7\gamma - k^{-0.4})\sqrt{k}/(1.1\sqrt{k}) \geq 0.8\alpha - 7\gamma - k^{-0.4}$, which is greater than $16\gamma + 3k^{-0.4}$ for our choice of $\alpha$. Hence, in this case, $T$ is not accepted w.r.t. $S$. It follows that the numerator of (5) is at most $e^{-\Omega(k^{0.1})}$. Since we also lowerbounded the denominator of (5) by $\epsilon/3$ and since $\epsilon > e^{-k^{0.001}}$, we obtain inequality (4), as required.

Using Claims 14 and 15, we immediately get the following.

**Claim 16.** With probability at least $(\epsilon/2)(1 - e^{-\Omega(k^{0.1})}) \geq \epsilon/3$, the set $S$ chosen in line 3 of our algorithm will have the following properties:

1. $S$ is $\gamma$-good,
2. all but $e^{-\Omega(k^{0.1})}$ fraction of $\gamma$-good sets $T$ are accepted w.r.t. $S$, and
3. all but $e^{-\Omega(k^{0.1})}$ fraction of sets $T$ accepted w.r.t. $S$ are $\alpha$-good, for $\alpha = 29\gamma + 6k^{-0.4}$.

We will continue the analysis of our algorithm, assuming that such a set $S$ satisfying properties (1)–(3) of Claim 16 was chosen. First we argue that the algorithm will reach line 11 quite often.

**Claim 17.** With probability at least $1 - o(1)$, the final value of the variable total of our algorithm is at least $\text{timeout} \times \epsilon/6 = 32k' \ln k'$.

*Proof.* By property (2) of the set $S$ and by the fact that there are at least $\epsilon/2$ fraction of $\gamma$-good sets $T$, we know that the probability of reaching line 11 in a single iteration is at least $\epsilon/2(1 - e^{-\Omega(k^{0.1})}) \geq \epsilon/3$. Thus, the expected value of total, which is the total number of times that line 11 was executed, is at least $\text{timeout} \times (\epsilon/3)$. By Chernoff, the probability that $\text{total} < \text{timeout} \times \epsilon/6$ is at most $e^{-\text{timeout} \times \epsilon/24} = e^{-8k' \ln k'}$. So, with high probability, the value total is at least $\text{timeout} \times \epsilon/6$. 

□
Now we bound the number of errors made by the algorithm in line 15.

**Claim 18.** With probability at least 1/2, the number of wrong outputs $output_i$ made in line 15 of the algorithm is at most $9\alpha k'$. 

**Proof.** By property (3) of the set $S$, we know that conditioned on extending one of the strings $Answers_i$ in line 11, the probability of extending that string with a wrong bit is at most $\alpha + e^{-\Omega(k^{0.1})} \leq 1.1\alpha$. Thus, the expected fraction of wrong bits in the entire collection of the strings $Answers_i$ will be at most $1.1\alpha$. Let $I$ denote the set of those $i \in [k']$ where $|Answers_i| > total/(2k')$ and Majority$Answers_i$ is wrong. Then the total fraction of wrong answers in all strings $Answers_i$ is at least $|I|/(4k')$. By the above, this is expected to be at most $1.1\alpha$. So, the expected number of those $i \in [k']$ where our algorithm makes a mistake in line 15 is at most $4.4\alpha k'$. Applying the Markov inequality, we get the required claim. \hfill $\square$

Next we argue that the number of $i \in [k']$ such that $output_i$ is set randomly in line 14 of the algorithm will be small. The idea is that when we extend the string $Answers_i$ in line 11 of the algorithm, we do it for an index $i \in [k']$ that is distributed almost uniformly over $[k']$. Since line 11 is reached many times, we can conclude that the fraction of indices $i \in [k']$ that were selected too few times will be small.

To argue that an index $i$ chosen in line 11 of the algorithm is distributed almost uniformly, we need the following variant of Lemma 11 for the case of sets rather than tuples, i.e., we now consider sampling without replacement.

**Lemma 19.** Let $S$ be the collection of all $m$-size sets $s \in [k]$, for any $m \leq k$. Let $G \subseteq S$ be any subset of $S$ that has weight at least $\epsilon$ under the uniform distribution over $S$. Let $U$ be the uniform distribution on elements $j \in [k]$, and let $D$ be the distribution defined as follows: pick a set $s \in G$ uniformly at random and output a random element of $s$. Then $\text{Dist}(D, U) \leq O\left(\sqrt{\frac{\log(m/\epsilon)}{m}}\right)$. 

We postpone the proof of Lemma 19 till the Appendix, and continue with the analysis of our direct product amplifying algorithm.

**Claim 20.** With probability at least $1 - o(1)$, the fraction of $i \in [k']$ such that $output_i$ will be randomly set in line 14 is at most $O(k^{-0.4})$. 

**Proof.** Given a sequence of $total$ guesses made by the algorithm, we will argue that most $i \in [k']$ have $|Answers_i| \geq total/(2k')$.

Recall that the collection of random sets $T$’s accepted with respect to $S$ has weight at least $\epsilon/3$ according to the uniform distribution over all size-$k$ random sets in $[k']$. By Lemma 19, we get that the distribution $D$ over $[k']$ induced by taking a random set $T$ accepted with respect to $S$ and then outputting a random element of $T$ has statistical distance at most $\beta \overset{\text{def}}{=} O\left(\sqrt{\frac{\log(k/\epsilon)}{k}}\right) \leq O(k^{-0.4})$ to the uniform distribution over $[k']$ (since $\epsilon > e^{-k^{0.001}}$).

By a Markov-style argument, the fraction of those $i \in [k']$ that have probability less that $3/(4k')$ under $D$ is at most $4\beta$. Let $J$ denote the set of the remaining $i$’s. For each $i \in J$, we expect the length of the string $Answers_i$ to be at least $\frac{3}{8}\frac{total}{k'}$. By the Chernoff bounds, the probability that $|Answers_i| < \frac{3}{8}\frac{total}{k'} < total/(2k')$ is at most $e^{-total/(16k')}$. By the union bound, the probability that there is at least one $i \in J$ with $|Answers_i| < total/(2k')$ is at most $k'e^{-total/(16k')}$. 

By Claim 17, the value $total \geq 32k' \ln k'$ with probability $1 - o(1)$. Conditioned on $total$ being that large, the probability of having at least one $i \in J$ set to a random bit in line 14 of the algorithm is at most $1/k'$. Thus, with probability $1 - o(1)$, only a set of size at most $4\beta$ of indices will be randomly set in line 14. The claim follows. \hfill $\square$
Now we can finish the analysis.

Proof of Lemma 12. With probability at least $\epsilon/2$, a random input $k'$-tuple $\bar{x}$ satisfies Eq. (2) of Claim 13. By Claim 16, with probability at least $\epsilon/3$ the set $S$ chosen in line 3 of the algorithm will satisfy the conclusions of Claim 16.

Conditioned on such $\bar{x}$ and $S$ being chosen, Claims 18 and 20 say that with probability at least $1/2 - o(1) > 1/4$, the output of our algorithm is correct on all but at most $O(\gamma + k^{-0.4})$ fraction of $i \in [k']$. Lifting the conditioning on $\bar{x}$ and $S$, we conclude that our algorithm $\epsilon' = (\epsilon^2/24)$-computes $(1 - O(\gamma + k^{-0.4}))$-Direct product $f^{k'}$. The running time bound is obvious. 

4.3 Analysis of algorithm LEARN

Here we will prove Lemma 9, which we re-state below. First we recall the definition of a $(\gamma, \kappa)$-good distribution. For parameters $k$, $n$, $t$, let $D$ be a probability distribution over $t$ blocks where each block consists of $k - 1$ pairs $(x, b_x)$, with $x \in \{0, 1\}^n$ and $b_x \in \{0, 1\}$. Let $D_x$ be the probability distribution over $\{0, 1\}^{n(k - 1)t}$ obtained from $D$ by keeping only $x$ from every pair $(x, b_x)$. For parameters $0 \leq \kappa, \gamma \leq 1$, we say that the distribution $D$ is $(\gamma, \kappa)$-good if

1. for every sample from $D$, each block $(x_1, b_1), \ldots, (x_{k - 1}, b_{k - 1})$ of the sample is such that $b_j = f(x_j)$ for at least $(1 - \gamma)$ fraction of $j \in [k - 1]$, and

2. the distribution $D_x$ is statistically $\kappa$-close to the uniform distribution over $\{0, 1\}^{n(k - 1)t}$.

Lemma 9 [Analysis of Algorithm LEARN]. For any $\mu \geq \nu > 0$, $\kappa > 0$, $\epsilon = e^{-k^{0.3}}$, $\rho = \log(1/\rho) + k^{-\nu/2}$, and $t = O((\log 1/\rho)/\epsilon^2)$, there is a probabilistic algorithm LEARN satisfying the following. Let $f$ be an $n$-variable Boolean function such that a circuit $C$ $\epsilon$-computes $(1 - k^{-\mu})$-Direct Product $f^k$, and let $D$ be a probability distribution over $(\{0, 1\}^n \times \{0, 1\})^{(k - 1)t}$ that is $(k^{-\nu}, \kappa)$-good. Then algorithm LEARN, given as input $C$ and a random sample from $D$, outputs with probability at least $1 - O(\rho) - \kappa$ a circuit $C'$ that computes $f$ on at least $1 - O(\rho)$ fraction of inputs. The running time of LEARN, and hence also the size of the circuit $C'$, is at most $\text{poly}(|C|, 1/\epsilon)$.

We recall the proof of the Direct Product Lemma from [IW97]. Given a circuit $C$ that computes the direct product function $f^k$ with probability at least $\epsilon$, consider the following distribution $F$ on randomized circuits $F$. On input $x$, pick $i \in [k]$ uniformly at random, and set $x_i = x$. For each $j \in [k] \setminus \{i\}$, get a sample $(x_j, f(x_j))$ where $x_j$ is uniformly distributed. Evaluate the circuit $C$ on the input $(x_1, \ldots, x_k)$. Let $z$ be the number of indices $j \in [k] \setminus \{i\}$ where the $j$th output of the circuit $C$ disagrees with the value $f(x_j)$. With probability $2^{-z}$, output the $i$th output of the circuit $C$, and with the remaining probability $1 - 2^{-z}$ output a random coin flip.

Impagliazzo and Wigderson [IW97] argue that, for every subset $H$ of at least $\delta$ fraction of inputs, a random circuit sampled according to $F$ will compute $f$ on a random input from $H$ with probability at least $1/2 + \Omega(\epsilon)$. Then they conclude that the circuit obtained by applying the majority function to a small number of sampled circuits from $F$ will compute $f$ on all but at most $\delta$ fraction of inputs.

We generalize the argument of [IW97] in two ways. First, we assume that the given circuit $C$ $\epsilon$-computes $(1 - \gamma)$-Direct Product $f^k$. Secondly, in the definition of the probability distribution $F$, instead of sampling $(k - 1)$-tuples $(x_j, f(x_j))$ for uniformly distributed $n$-bit strings $x_j$’s, we will use the samples that come from a $(k^{-\nu}, \kappa)$-good distribution $D$.

Our modification of the analysis from [IW97] will be as follows. After computing the number $z$ of mistakes the circuit makes on inputs coming from the samples $(x, b_x)$, we will subtract from $z$ $\gamma k$ errors that a circuit computing the $(1 - \gamma)$-Direct Product $f^k$ can make (even for the “good”
\(k\)-tuples) as well as \(k^{-\nu}(k-1)\) errors that can be present in our imperfect sample coming from the distribution \(D\). Let \(w = z - \gamma k - k^{-\nu}(k-1)\). We will decide with probability \(\lambda^w\) to believe the output of the circuit on our input of interest, and with the remaining probability we use a fair coin flip as our prediction for the value of \(f\). The choice of the parameter \(\lambda\) will now depend on the amount of extra errors we have to deal with. It turns out sufficient to set \(\lambda = (1/2)^{1/(k^{-\nu}(k-1))}\).

More formally, we have the following lemma for the case \((k^{-\nu}, 0)\)-good distribution \(D\).

**Lemma 21.** For \(\mu \geq \nu > 0\) and \(\epsilon > e^{-k^{7/3}}\), let \(f\) be an \(n\)-variable Boolean function such that a circuit \(C\) \(\epsilon\)-computes \((1 - k^{-\nu})\)-Direct Product \(f^k\). Let \(D\) be a distribution on \((k-1)\)-tuples \((x_1, b_1), \ldots, (x_{k-1}, b_{k-1})\) such that

1. the \(x_j\)s are independent uniformly distributed random variables over \(\{0, 1\}^n\), and
2. for every sample from \(D\), we have \(b_j = f(x_j)\) for at least \((1 - k^{-\nu})\) fraction of \(j \in [k-1]\).

Then there is a probability distribution \(\mathcal{F}\) over randomized Boolean circuits \(F\) such that, for every set \(H \subseteq \{0, 1\}^n\) of density at least \(\delta = k^{-\nu}/2 + \frac{8\ln(100/\epsilon)}{k}\),

\[
\Pr_{F \in \mathcal{F}, x \sim H}[F(x) = f(x)] > 1/2 + \Omega(\epsilon).
\]

Moreover, \(\mathcal{F}\) is sampleable in time \(\text{poly}(1/\epsilon, |C|)\) given input \(C\) and one sample from \(D\).

**Proof.** It will be convenient for us to view the distribution \(D\) as a pair of distributions \((U, B|U)\), where \(U\) is uniform over \(\{0, 1\}^{n(k-1)}\) (think of them as \(k-1\) strings \(y_1, \ldots, y_{k-1}\)) and \(B|U\) is the distribution on \(\{0, 1\}^{k-1}\) conditioned on \(U\). So sampling \((y_1, b_1), \ldots, (y_{k-1}, b_{k-1})\) from \(D\) is equivalent to sampling \(y_1, \ldots, y_{k-1}\) uniformly at random, and then sampling \(b_1, \ldots, b_{k-1}\) from \(B\) conditioned on \(y_1, \ldots, y_{k-1}\). By our assumption, every sample from \(B\) has the property that \(b_i = f(y_i)\) for all but at most \(k^{-\nu}\) fraction of positions \(i \in [k-1]\).

Let \(\gamma = k^{-\mu}\) and \(\gamma' = k^{-\nu}\). Consider the following distribution \(\mathcal{F}\) on randomized circuits \(F\):

“On input \(x\),

1. sample \(i \in [k]\) uniformly at random, and set \(x_i = x\);
2. sample \(y_1, \ldots, y_{k-1}\) from \(U\), and assign the values \(y_1, \ldots, y_{k-1}\) to \(x_j\) for \(j \in [k] \setminus \{i\}\);
3. evaluate the circuit \(C\) on the input \((x_1, \ldots, x_k)\);
4. sample \(b_1, \ldots, b_{k-1}\) from the distribution \(B\) conditioned on the \(x_j\)s for \(j \in [k] \setminus \{i\}\);
5. let \(z\) be the number of indices \(j \in [k] \setminus \{i\}\) where the \(j\)th output of the circuit \(C\) disagrees with the corresponding value \(b_j\); let \(t = \gamma'(k-1) + \gamma k\). If \(z \leq t\), then output the \(i\)th output of the circuit \(C\). Otherwise, for \(w = z - t\) and \(\lambda = 2^{-1/\gamma'(k-1)}\), with probability \(\lambda^w\) output the \(i\)th output of the circuit \(C\), and with the remaining probability output a random coin flip.”

Let \(H\) be any set of density \(\delta\). Suppose we pick \(x \in H\) uniformly at random, and then sample a random circuit \(F\) according to the distribution \(\mathcal{F}\) defined above. The random choice of \(x \in H\) and random choices of the first two steps of the algorithm for \(F\) induce a probability distribution \(\mathcal{E}\) on \((i; x_1, \ldots, x_k)\). Let \(\mathcal{E}'\) be the marginal probability distribution on \((x_1, \ldots, x_k)\) induced by \(\mathcal{E}\). Note that in step 3 of the algorithm described above, we run the circuit \(C\) exactly on those \(k\)-tuples that come from the distribution \(\mathcal{E}'\).

We have the following.
Claim 22. The probability distribution $\mathcal{E}$ is equivalent to the following distribution: sample a $k$-tuple $(x_1, \ldots, x_k)$ according to $\mathcal{E}'$, sample $i \in [k]$ uniformly at random among the positions that contain strings from $H$, and output $(i; x_1, \ldots, x_k)$.

Proof. Let $N = 2^n$. A string $(i; x_1, \ldots, x_k)$, with $x_i \in H$, has probability exactly $\frac{1}{k} \frac{1}{H|Nk^{k-1}}$ according to $\mathcal{E}$. On the other hand, $\mathcal{E}'$ assigns to the string $x_1, \ldots, x_k$ the probability $h \frac{1}{k} \frac{1}{H|Nk^{k-1}}$, where $h$ is the number of positions $j \in [k]$ such that $x_j \in H$. It follows that for given a sample from $\mathcal{E}'$, the value of $i$ is distributed uniformly over the $h$ positions containing strings from $H$.

Thanks to Claim 22, we can equivalently view the distribution $\mathcal{F}$ of circuits $F$ on random inputs from $H$ as follows:

1. sample $(x_1, \ldots, x_k)$ from $\mathcal{E}'$;
2. evaluate the circuit $C$ on the input $(x_1, \ldots, x_k)$;
3. sample $i \in [k]$ uniformly at random among those $j$ where $x_j \in H$;
4. sample $b_1, \ldots, b_{k-1}$ from the distribution $B$ conditioned on the $x_j$s for $j \in [k] \setminus \{i\}$;
5. let $z$ be the number of indices $j \in [k] \setminus \{i\}$ where the $j$th output of the circuit $C$ disagrees with the corresponding value $b_j$; let $t = \gamma'(k-1) + \gamma k$. If $z \leq t$, then output the $t$th output of the circuit $C$. Otherwise, for $w = z - t$ and $\lambda = 2^{-1/\gamma'(k-1)}$, with probability $\lambda^w$ output the $i$th output of the circuit $C$, and with the remaining probability output a random coin flip.

In the rest of the proof, we will use the given equivalent description of $\mathcal{F}$ on a random input from $H$ to estimate the probability that the answer obtained in step 5 is correct.

The next two claims show that $\mathcal{E}'$ behaves very similarly to the uniform distribution.

Claim 23. $\Pr_{\bar{x}=(x_1, \ldots, x_k)\sim \mathcal{E}'}[\bar{x}$ contains fewer than $\delta k/2$ elements from $H] \leq \epsilon/200$.

Proof. Observe that a $k$-tuple that contains exactly $s$ strings from $H$ is assigned in $\mathcal{E}'$ the probability $\frac{s}{k} \frac{1}{H|Nk^{k-1}}$, which is exactly $\frac{s}{\delta k}$ times the probability of this $k$-tuple under the uniform distribution. For a random $k$-tuple of $n$-bit strings, the expected number of strings falling in the set $H$ is $\delta k$. By Chernoff, the fraction of those $k$-tuples that contain fewer than $\delta k/2$ strings from $H$ is at most $e^{-\delta k/8}$. By our choice of $\delta$, this is at most $\epsilon/100$. Since the collection of $k$-tuples that contain fewer than $\delta k/2$ strings from $H$ is assigned by $\mathcal{E}'$ the weight at most $1/2$ of their weight under the uniform distribution, the claim follows.

Claim 24. Distribution $\mathcal{E}'$ assigns weight at least $0.49\epsilon$ to the collection of $k$-tuples $(x_1, \ldots, x_k)$ such that they contain at least $\delta k/2$ elements from $H$ and $C(x_1, \ldots, x_k)$ and $f^k(x_1, \ldots, x_k)$ agree in at least $(1-\gamma)$ fraction of places.

Proof. We know that $C$ does well on at least $\epsilon$ fraction of $k$-tuples under the uniform distribution. By Chernoff, all but $\epsilon/100$ fraction of these tuples will have fewer than $\delta k/2$ strings from $H$. So, under the uniform distribution, for at least $0.99\epsilon$ fraction of $k$-tuples we have that $C$ is correct on at least $(1-\gamma)$ fraction of places, and that the tuple contains at least $\delta k/2$ strings in $H$. The latter implies that the collection of such tuples gets in $\mathcal{E}'$ at least $1/2$ of their probability mass under the uniform distribution, which is at least $(1/2)0.99\epsilon \geq 0.49\epsilon$. 

22
We now estimate the probability of getting a correct answer in step 5 of our algorithm for a random $k$-tuple from $E'$. We will only consider those $k$-tuples that contain at least $\delta k/2$ elements from $H$. By Claim 23, this may introduce an additive error of at most $\epsilon/200$.

We divide the $k$-tuples containing at least $\delta k/2$ of strings from $H$ into two sets: $G_{few}$ containing $k$-tuples where $C$ is wrong in at most $\gamma k$ positions, and $G_{many}$ containing the remaining $k$-tuples. We estimate the success probability of our algorithm separately on $G_{few}$ and $G_{many}$. In fact, it will be more convenient for us to estimate the advantage of our algorithm, i.e., the probability of getting a correct answer in step 5 minus the probability of getting a wrong answer.

**Claim 25.** For every $k$-tuple from $G_{few}$, the advantage of our algorithm is at least 0.9.

**Proof.** Suppose that $\bar{x}$ is a $k$-tuple from $G_{few}$ which contains $h \geq \delta k/2$ strings in $H$. Suppose that $r \leq \gamma k$ is the total number of mistakes the circuit $C$ makes on this tuple, and $l \leq r$ is the number of mistakes $C$ makes on elements from $H$ in the tuple.

Let $i \in [k]$ be uniformly chosen among the $h$ positions that contain a string in $H$. By the assumption on the probability distribution $B$, we know that the number $z$ of disagreements between the $j$th outputs of $C$, for $j \neq i$, and the corresponding bits $b_j$ supplied by $B$ is at most $\gamma/(k-1)+\gamma k$. Hence, in step 5 we will output the $i$th output of $C$ with probability 1. The probability (over the choice of $i$) that this output is wrong is $l/h$, and so the advantage is $1 - 2l/h$. Note that $l/h \leq 2\gamma/\delta \leq 2k^{-\mu}/k^{-\nu/2} \leq 2/k^{\mu/2} \leq o(1)$. Hence, the advantage is at least $1 - o(1) \geq 0.9$, as claimed. 

**Claim 26.** For every $k$-tuple from $G_{many}$, the advantage of our algorithm is at least $-2^{-\Omega(k^{1/2})}$.

**Proof.** Suppose a given $k$-tuple $\bar{x}$ from $G_{many}$ contains $h \geq \delta k/2$ elements from $H$. Suppose that $C$ is incorrect in $r > \gamma k$ positions, and $l \leq r$ is the number of mistakes $C$ makes in positions containing strings from $H$.

Let $i \in [k]$ be uniformly chosen among the $h$ positions that contain a string in $H$. Let $z$ denote the number of disagreements between the $j$th outputs of $C$, for $j \neq i$, and the corresponding bits $b_j$ supplied by $B$. By our assumption on the distribution $B$, we have

$$r - 1 - \gamma/(k - 1) \leq z \leq r + \gamma/(k - 1)$$

for every choice of $i$. So for $w = z - t$, we have

$$r - 1 - 2\gamma/(k - 1) - \gamma k \leq w \leq r - \gamma k.$$

With probability $1 - l/h$, the index $i$ is such that $C$ is correct on $x_i$. In this case, the advantage of our algorithm is

$$\lambda^w \geq \lambda^{r - \gamma k}$$

(since $\lambda \leq 1$). With probability $l/h$, the index $i$ is such that $C$ is wrong on $x_i$. In this case, the advantage is

$$-\lambda^w \geq -\lambda^{r - 1 - 2\gamma/(k - 1) - \gamma k}.$$

So overall, the advantage is at least

$$(1 - l/h)\lambda^{r - \gamma k} - (l/h)\lambda^{r - 1 - 2\gamma/(k - 1) - \gamma k}. \quad (11)$$

To lowerbound the quantity in Eq. (11), we consider two cases.
• **Case 1:** \( r - \gamma k \leq 2\gamma'(k - 1) + 1 \). Then the quantity in Eq. (11) is at least

\[
(1 - l/h)\lambda^{2\gamma'(k-1)+1} - (l/h) \geq (1 - l/h)\lambda^{3\gamma'(k-1)} - (l/h),
\]

which is at least \( 1/8 - 9/8(l/h) \), since \( \lambda^{\gamma'(k-1)} = 1/2 \). Note that \( l \leq r \leq 2\gamma'(k - 1) + 1 + \gamma k \) (due to the assumption of Case 1). Also recall that \( h \geq \delta k/2 > \sqrt{\gamma k}/2 \). Hence, \( l/h \leq (2\gamma' + \gamma + 1/k)/(\sqrt{\gamma}/2) \leq (3\gamma' + 1/k)/(\sqrt{\gamma}/2) \leq o(1) \). It follows that the quantity in Eq. (11) is at least \( 1/8 - o(1) > 0 \) in this case.

• **Case 2:** \( r - \gamma k > 2\gamma'(k - 1) + 1 \). Then the quantity in Eq. (11) is at least

\[
\lambda^{r-\gamma k}((1 - l/h) - (l/h)\lambda^{-3\gamma'(k-1)}),
\]

where we upperbounded \( 2\gamma'(k - 1) + 1 \) by \( 3\gamma'(k - 1) \) (which is correct for sufficiently large \( k \)). Since \( \lambda^{\gamma'(k-1)} = 1/2 \), the expression in Eq. (12) is at least \( \lambda^{r-\gamma k}(1 - 9(l/h)) \).

If \( l < h/9 \), this expression is positive. If \( l \geq h/9 \), this expression is at least

\[
-8\lambda^{r-\gamma k} \geq -8\lambda^{l-\gamma k} \geq -8\lambda^{h/9 - \gamma k} \geq -8\lambda^{\delta k/18 - \gamma k} \geq -8\lambda^{\sqrt{\gamma k}/18 - \gamma' k}.
\]

This can be lowerbounded by \(-8(1/2)^{(1/(18\sqrt{\gamma}))} \), again using the fact that \( \lambda^{\gamma'(k-1)} = 1/2 \). Recalling that \( \gamma' = k/\nu' \), we conclude that the advantage in this case is at least \(-2^{-\Omega(k^{k/2})} \).

So in both cases, the quantity in Eq. (11) is at least \(-2^{-\Omega(k^{k/2})} \), as claimed.

Finally, we have by Claims 24 and 25 that the advantage due to tuples in \( G_{\text{few}} \) is at least \((0.49)e(0.9) > 0.44\epsilon \). By Claim 26, the advantage due to tuples in \( G_{\text{many}} \) is at least \(-2^{-\Omega(k^{k/2})} \). By Claim 23, the advantage due to tuples outside of \( G_{\text{few}} \cup G_{\text{many}} \) is at least \(-\epsilon/200 \). Thus the overall advantage is at least \( 0.44\epsilon - 2^{-\Omega(k^{k/2})} - \epsilon/200 \), which is at least \( 0.43\epsilon \) when \( \epsilon > e^{-k^{k/3}} \). This means that the success probability of a random circuit \( F \) from \( \mathcal{F} \) on a random \( x \in H \) is at least \( 1/2 + 0.2\epsilon \), as claimed.

**Proof of Lemma 9.** First we argue the case of \( \kappa = 0 \). By Lemma 21, we get for a sufficiently small constant \( \alpha > 0 \) that the set \( \text{Bad} = \{ x \in \{0,1\}^n \mid \Pr_{F \leftarrow \mathcal{F}}[F(x) = f(x)] \leq 1/2 + \alpha \epsilon \} \) must have density less than \( O(\rho) \).

Consider any fixed string \( x \notin \text{Bad} \). Let \( M_t \) denote the Majority circuit applied to \( t \) random circuits sampled from \( \mathcal{F} \) on input \( x \). By Chernoff, a random circuit \( M_t \) is correct on \( x \) with probability at least \( 1 - \rho^2 \). By averaging, we get that for at least \( 1 - \rho \) fraction of circuits \( M_t \), it is the case that \( M_t \) is correct on at least \( 1 - \rho \) fraction of inputs \( x \notin \text{Bad} \). It follows that if we pick one such circuit \( M_t \) at random, then with probability at least \( 1 - \rho \) it will be correct on all but \( \rho \) fraction of inputs \( x \notin \text{Bad} \). So it will compute the function \( f \) on at least \( 1 - O(\rho) \) fraction of all inputs.

For \( \kappa \neq 0 \), we will argue that the probability that the algorithm described in the previous paragraph succeeds in finding a good circuit for \( f \) changes by at most \( \kappa \). Let us suppose, for contradiction, that this success probability changes by more than \( \kappa \). The distribution \( D \) can be viewed as a pair \((D', D'')\), where \( D'' \) is a distribution on \( t \) blocks of \( (k - 1) \)-tuples \( (b_1, \ldots, b_{k-1}) \), for \( b_j \in \{0,1\} \), conditioned on a given sample from \( D' \). Then the following is a statistical test distinguishing between \( D' \) and the uniform distribution: Given a sample \( a \) of \( t \) blocks of \( (k - 1) \)-tuples \( x_1, \ldots, x_{k-1} \), sample from \( D'' \) conditioned on the given sample \( a \). (If the sample \( a \) has zero probability under \( D' \), then we know that it came from the uniform distribution. So we may assume,
without loss of generality, that \( a \) has nonzero probability in \( D' \).) Then run the algorithm described above that constructs a circuit for the function \( f \). If the constructed circuit agrees with \( f \) on at least \( 1 - O(\rho) \) fraction of inputs, then accept; otherwise reject.

It follows that the described test accepts uniform \( a \) with probability at least \( 1 - \rho \), while it accepts \( a \) from \( D' \) with probability less than \( 1 - \rho - \kappa \). Hence the two distributions have statistical distance more than \( \kappa \). A contradiction.

\[\square\]

5 Applications

5.1 Local approximate list-decoding of truncated Hadamard codes

Recall the definition of truncated Hadamard codes. For given \( n, k \in \mathbb{N} \), a \( k \)-truncated Hadamard encoding of a message \( \text{msg} \in \{0, 1\}^n \) is defined as a string \( \text{cmsg} \in \{0, 1\}^{\binom{n}{k}} \), where the codeword is indexed by \( k \)-sets \( s \subseteq [n], |s| = k \), and \( \text{cmsg}(s) = \oplus_{i \in s} \text{msg}(i) \). Using our local approximate list-decoding algorithm for direct product codes in Theorem 1 and the list-decoding algorithm for Hadamard codes of Goldreich and Levin \([GL89]\), we get an efficient approximate list-decoding algorithm for \( k \)-truncated Hadamard codes.

**Theorem 27.** There is a randomized algorithm \( A \) with the following property. Let \( \text{msg} \) be any \( N = 2^n \)-bit string. Suppose that the truth table of the Boolean function computed by a circuit \( C \) agrees with the \( k \)-truncated Hadamard encoding \( \text{cmsg} \) of \( \text{msg} \) in at least \( 1/2 + \epsilon \) fraction of positions, where \( \epsilon = \Omega(\text{poly}(1/k)) \). Then the algorithm \( A \), given \( C \), outputs with high probability a list of at most \( \text{poly}(1/\epsilon) \) Boolean circuits \( C' \) such that there is at least one \( C' \) on the list that agrees with \( \text{msg} \) in at least \( 1 - \rho \) fraction of positions, where \( \rho = O(k^{-0.1}) \). The running time of the algorithm \( A \), and hence also the size of \( C' \), is at most \( \text{poly}(|C|, 1/\epsilon) \).

First we prove a more uniform version of Yao’s XOR Lemma. Recall that a Boolean function \( f \) is \( \delta \)-hard with respect to circuit size \( s \) if, for every Boolean circuit \( C \) of size at most \( s \), \( C \) correctly computes \( f \) on at most \( 1 - \delta \) fraction of inputs. Yao’s XOR Lemma \([Yao82]\) says that if a Boolean function \( f \) is \( \delta \)-hard with respect to circuit size \( s \), then the function \( f^{\otimes k}(x_1, \ldots, x_k) = \oplus_{i=1}^k f(x_i) \) is \( 1/2 - \epsilon \)-hard with respect to circuit size \( s' \), where \( \epsilon \approx (1 - \delta)^k/2 \) and \( s' = s + \text{poly}(\epsilon, \delta) \).

It is easy to prove Yao’s XOR Lemma given the Direct Product Lemma, using the result of Goldreich and Levin \([GL89]\) on list-decoding Hadamard codes. Applying this to our version of the Direct Product Lemma (Theorem 1), we prove the following version of the XOR Lemma.

**Lemma 28 (Advice-efficient XOR Lemma).** There is a randomized algorithm \( A \) with the following property. Let \( f \) be any \( n \)-variable Boolean function. Suppose a circuit \( C \) computes the function \( f^{\otimes k}(x_1, \ldots, x_k) \) on at least \( 1/2 + \epsilon \) fraction of inputs, where each \( x_i \in \{0, 1\}^n \), and \( \epsilon = \Omega(\text{poly}(1/k)) \). Then the algorithm \( A \), given \( C \), outputs with probability at least \( \varepsilon' = \text{poly}(\epsilon) \) a circuit \( C' \) such that \( C' \) agrees with \( f \) on at least \( 1 - \rho \) fraction of inputs, where \( \rho = O(k^{-0.1}) \). The running time of the algorithm \( A \), and hence also the size of \( C' \), is at most \( \text{poly}(|C|, 1/\epsilon) \).

For the proof, we will need the following result due to Goldreich and Levin.

**Lemma 29 ([GL89]).** There is a probabilistic algorithm \( A \) with the following property. Let \( x \in \{0, 1\}^n \) be any string, and let \( B : \{0, 1\}^n \rightarrow \{0, 1\} \) be any predicate such that \( \Pr_{r \in \{0, 1\}^n}[B(r) = \langle x, r \rangle] \geq 1/2 + \gamma \), for some \( \gamma > 0 \). Then, given oracle access to \( B \), the algorithm \( A \) runs in time \( \text{poly}(n, 1/\gamma) \), and outputs the string \( x \) with probability at least \( \Omega(\gamma^2) \).
Proof of Lemma 28. Consider the function $F : \{0,1\}^{2nk} \times \{0,1\}^{2k} \to \{0,1\}$ defined as follows: For $x_1, \ldots, x_{2k} \in \{0,1\}^n$ and $r \in \{0,1\}^{2k}$,

$$F(x_1, \ldots, x_{2k}, r) = \langle f(x_1) \ldots f(x_{2k}), r \rangle.$$ 

Note that conditioned on $r \in \{0,1\}^{2k}$ having exactly $k$ 1s, the function $F(x_1, \ldots, x_{2k}, r)$ is distributed exactly like the function $f^{\oplus k}(x_1, \ldots, x_k)$, for uniformly and independently chosen $x_i$s.

Consider the following algorithm for computing $F$. Given an input $x_1, \ldots, x_{2k}, r$, count the number of 1s in the string $r$. If it is not equal to $k$, then output a random coin flip and halt. Otherwise, simulate the circuit $C$ on the sub-tuple of $x_1, \ldots, x_{2k}$ of size $k$ which is obtained by restricting $x_1, \ldots, x_{2k}$ to the positions in $r$ that are 1, and output the answer of $C$.

Let $p$ be the probability that a random $2k$-bit string contains exactly $k$ 1s. It is easy to see that the described algorithm for computing $F$ is correct with probability at least $(1-p)/2 + p(1/2 + \varepsilon) = 1/2 + p\varepsilon$. Since $p \geq \Omega(1/\sqrt{k})$, we get that our algorithm for $F$ is correct with probability at least $1/2 + \varepsilon'$, for $\varepsilon' = \Omega(\varepsilon/\sqrt{k})$.

By a Markov-style argument, we have that for each of at least $\varepsilon'' = \varepsilon'/2$ of the $2k$-tuples $x_1, \ldots, x_{2k}$, our algorithm computes $F(x_1, \ldots, x_{2k}, r)$ for at least $1/2 + \varepsilon''$ fraction of $r$s. Applying the Goldreich-Levin algorithm of Lemma 29 gives us an algorithm for computing the Direct Product function $f^{2k}(x_1, \ldots, x_{2k})$ for at least $\Omega(\varepsilon''^3)$ fraction of inputs. Finally, applying the algorithm of Theorem 1 to this Direct Product algorithm yields, with probability $\text{poly}(\varepsilon)$, a circuit computing $f$ on at least $1 - \rho$ fraction of inputs of $f$, as required.

Observe that Lemma 28 gives us an algorithm for decoding a version of truncated Hadamard codes where, instead of sets of size $k$, the codeword is indexed by tuples of size $k$. Moreover, the decoding is **local** in that, once we compute the list of circuits for the original Boolean function (whose truth table is viewed as the message), we can run each of these circuits on a given input $x$ to produce the $x$th bit of the original message. Using Lemma 28, we also get an approximate list-decoding algorithm for the original version of truncated Hadamard codes.

Proof of Theorem 27. The proof is by a reduction to Lemma 28. Let $f : \{0,1\}^n \to \{0,1\}$ be the Boolean function with the truth table $\text{msg}$. Given the circuit $C$ from the statement of the theorem, we construct the following new circuit $C''$ for computing the XOR function $f^{\oplus k}$: Given a $k$-tuple $(x_1, \ldots, x_k)$ of $n$-bit strings, check if they are all distinct. If so, then run the circuit $C$ on the set $\{x_1, \ldots, x_k\}$, outputting the answer of $C$. Otherwise, output a random coin flip.

Let $p$ be the probability that a random $k$-tuple of $n$-bit strings contains more than one occurrence of some string. Then the described circuit $C''$ is correct on at least $p/2 + (1-p)(1/2 + \varepsilon) = 1/2 + (1-p)\varepsilon$ fraction of inputs. For $k = \text{poly}(n)$, it is easy to upperbound $p$ by $2^{-\Omega(n)}$. So the algorithm $C''$ is correct on at least $1/2 + \varepsilon/2$ fraction of inputs.

Running the algorithm of Lemma 28 on the constructed circuit $C''$ gives us, with probability at least $\text{poly}(\varepsilon)$, a circuit $C'$ that $(1-\rho)$-computes the function $f$. Repeatedly running this algorithm for $\text{poly}(1/\varepsilon)$ times would give us a list of circuits such that, with probability exponentially close to 1, at least one of the circuits on the list $(1-\rho)$-computes the function $f$. \hfill \qed

5.2 Uniform hardness amplification in $\mathcal{BPP}$

Here we will prove Theorem 2. First we recall some definitions. We say that a Boolean function family $f$ is $\delta$-hard with respect to probabilistic polynomial-time algorithms if any such algorithm computes $f$ correctly on at most $1-\delta$ fraction of $n$-bit inputs, where $\delta$ is some function of the input
size $n$. Similarly we can define hardness with respect to probabilistic polynomial-time algorithms using advice. We use the model of probabilistic algorithms taking advice as defined in [TV02]: the advice may depend on the internal randomness of the algorithm but is independent of the given input.

Our Theorem 27 (or rather its version for the XOR Lemma, Lemma 28) immediately gives us hardness amplification for probabilistic algorithms with small amount of advice.

Lemma 30. Suppose $f: \{0, 1\}^n \rightarrow \{0, 1\}$ is a Boolean function family such that, for some constant $c$, $f$ is $1/n^c$-hard with respect to any probabilistic polynomial-time algorithm with $O(\log n)$-size (randomness dependent) advice. Then the function $f^{\oplus k}$, for $k = n^{1+c}$, cannot be $(1/2 + 1/n^d)$-computed by any probabilistic polynomial-time algorithm for any $d$.

First, we observe that our Lemma 28 immediately gives us hardness amplification in the nonuniform setting with very small amount of advice.

Lemma 31. Let $f: \{0, 1\}^n \rightarrow \{0, 1\}$ be a Boolean function family. If there is a probabilistic polynomial-time algorithm that agrees with the function $f^{\oplus k}$ on at least $1/2 + \epsilon$ fraction of inputs, for some $k = \text{poly}(n)$ and $\epsilon = \text{poly}(1/k)$, then there is a probabilistic polynomial-time algorithm that, given advice of size $O(\log 1/\epsilon)$, agrees with $f$ on at least $1 - \delta$ fraction of inputs, for $\delta \leq O(k^{-0.1})$.

Proof. On an input $x$, we do the following. Given a probabilistic algorithm $(1/2 + \epsilon)$-computing the XOR function $f^{\oplus k}$, we apply to it the algorithm of Lemma 28 for some $\text{poly}(1/\epsilon)$ number of attempts. This gives us a list of $\text{poly}(1/\epsilon)$ Boolean circuits such that, with probability exponentially close to 1, at least one of the circuits on the list $(1 - \delta)$-computes $f$. The advice string of size $\log \text{poly}(1/\epsilon) = O(\log 1/\epsilon)$ can then be used to identify the correct circuit on the list. We output the result of running this circuit on the input $x$. \hfill \square

Now Lemma 30 is an immediate corollary of Lemma 31 above.

Logarithmic advice can sometimes be eliminated. For functions in NP, we can use the average-case search-to-decision reduction due to Ben-David, Chor, Goldreich, and Luby [BDCGL92] to obtain the following lemma.

Lemma 32. Suppose there is a language $L \in \text{NP}$ and a constant $c$ such that $L$ is $1/n^c$-hard with respect to probabilistic polynomial-time algorithms. Then there is a language $L' \in \text{NP}$ and a constant $d$ such that $L'$ is $1/n^d$-hard with respect to probabilistic polynomial-time algorithm taking $O(\log n)$ bits of advice.

We will need the average-case “search-to-decision” reduction for NP from [BDCGL92]; we state this result in the form it was stated in [Tre05].

Lemma 33 ([BDCGL92]). Let $L \in \text{NP}$ be any language, and let $R(\cdot,\cdot)$ be the polynomial-time relation defining $L$, where $|y| \leq w(|x|)$ for some polynomial function $w(\cdot)$. Then there exist a language $L' \in \text{NP}$, a polynomial $l(\cdot)$, and a probabilistic polynomial-time algorithm $A$ such that the following holds. Given a circuit $C'$ that $(1-\delta')$-computes $L'$ on inputs of length $l(n)$, the algorithm $A$ outputs, with probability at least $1 - 2^{-\text{poly}(n)}$, a circuit $C$ that solves the search version of $L$ (with respect to $R$) on at least $1 - \delta$ fraction of inputs of size $n$, for $\delta = O(\delta' w^2(n))$.

Proof of Lemma 32. Let $L$ be a given $\delta = 1/n^c$-hard language in NP, let $R$ be its defining relation, and let $w(n) = n^a$ be the upper-bound on the witness size for $n$-bit inputs in $L$. We claim that the language $L' \in \text{NP}$ given in Lemma 33 is $\delta' = \Omega(\delta/w^2(n))$-hard, on $l(n)$-bit inputs, with respect to probabilistic polynomial-time algorithms with $O(\log n)$ bits of advice.
Indeed, suppose there is an advice-taking probabilistic polynomial-time algorithm that \((1 - \delta')\)-computes \(L'\). Enumerate all polynomially many advice strings used by this algorithm, getting a list of polynomially many circuits such that, with probability exponentially close to 1, at least one of the circuits on the list will \((1 - \delta')\)-compute \(L'\). Apply the probabilistic polynomial-time algorithm \(A\) of Lemma 33 to each of the circuits on the list. This yields a list of circuits such that, with high probability, at least one of them solves the search version of \(L\) (with respect to \(R\)) for at least \(1 - \delta\) fraction of inputs. Simulate each of these circuits on a given input \(x\), and accept iff at least one of them produces a witness \(y\) such that \(R(x, y)\) holds. It follows that we have a probabilistic polynomial-time algorithm \((1 - \delta)\)-computing \(L\), contradicting the assumed hardness of \(L\). \(\square\)

Combining Lemma 30 and Lemma 32, we obtain the following.

**Theorem 34.** Suppose there is a Boolean function family \(f \in \NP\) and a constant \(c\) such that \(f\) is \(1/n^c\)-hard with respect to probabilistic polynomial-time algorithms. Then there is a Boolean function family \(g \in \P^{\NP\|}\) that cannot be computed by any probabilistic polynomial-time algorithm on more than \(1/2 + 1/n^d\) fraction of inputs, for any constant \(d\).

Finally, we observe that the existence of a hard function in \(\P^{\NP\|}\) implies the existence of a hard function in \(\NP\), and so Theorem 34 can be used to achieve uniform hardness amplification in \(\P^{\NP\|}\) claimed in Theorem 2.

**Proof of Theorem 2.** Let \(f\) be computed by a SAT-oracle Turing machine \(M\) in time \(n^e\), for some constant \(e\). Define a new Boolean function \(h\) as follows: For \(x \in \{0, 1\}^n\) and \(i \in [n^c]\), \(h(x, i) = 1\) iff \(M\) on input \(x\) makes at least \(i\) SAT-oracle queries and the \(i\)th SAT-oracle query is a satisfiable formula.

Clearly, the function \(h\) is in \(\NP\): Given \(x\) and \(i\), we simulate the oracle machine \(M\) on \(x\), recording all SAT-oracle queries asked by \(M\) — this can be done since \(M\) asks all its queries in parallel. If the number of queries is less than \(i\), we reject. Otherwise, we nondeterministically check that the \(i\)th SAT-oracle query is a satisfiable formula.

Next we argue that the function \(h\) is at least \(1/(n^c n^e)\)-hard with respect to probabilistic polynomial-time algorithms. Indeed, suppose this is not the case. Then there is a probabilistic polynomial-time algorithm \(A\) such that for each of at least \(1 - 1/n^c\) fraction of \(xs\) the algorithm \(A\) errs on less than \(1/n^e\) fraction of inputs \((x, i)\). Since, for each such \(x\), the number of different inputs \((x, i)\) is at most \(n^e\), we conclude that \(A\) is correct on all SAT-oracle queries made by the machine \(M\) on input \(x\). Hence, using this algorithm \(A\) to answer SAT-oracle queries, we get a probabilistic polynomial-time algorithm \((1 - 1/n^c)\)-computing \(f\), which contradicts the assumed hardness of \(f\).

Applying Theorem 34 to the \(\NP\)-function \(h\), we get the required hard function \(g \in \P^{\NP\|}\). \(\square\)

Trevisan [Tre05] gives uniform hardness amplification for \(\NP\): If \(\NP\) contains a language that is \(1/\poly(n)\)-hard with respect to probabilistic polynomial-time algorithms, then \(\NP\) contains a language that is \((1/2 - 1/\log^\alpha n)\)-hard, for some constant \(\alpha\). Our Theorem 2 achieves much better hardness amplification: from \(1/\poly(n)\)-hardness to \((1/2 - 1/n^d)\)-hardness for any \(d\). However, it only applies to the class \(\P^{\NP\|}\) rather than \(\NP\).

**Remark 35.** It is also possible to prove a “scaled-up” version of Theorem 2 where the hardness is against \(2^{2^{\alpha(1)}}\)-time probabilistic algorithms, and the amplification goes from hardness \(1/n^c\) to \(1/2 - 2^{-n^{\alpha(1)}}\).
6 Concluding remarks

The direct product is a generic construction widely used in complexity theory and coding theory. For instance, Alon et al. [ABN+92] used (a derandomized version of) the direct product construction to obtain error-correcting codes with large distance from codes with small distance. The simplicity of this purely combinatorial construction allowed for the subsequent development of linear-time encodable and list-decodable codes [GI03]; this kind of efficiency appears to be impossible to achieve using algebra-based codes. The direct product construction on graphs was used by Reingold, Vadhan, and Wigderson [RVW02] to obtain large expander graphs from smaller ones; this simple graph operation was then combined with an ingenious method to reduce the degree of the new graph while preserving its expansion properties, leading to a beautiful combinatorial construction of arbitrarily large expanders. A derandomized direct product construction similar to that of [ABN+92] was used in a recent breakthrough result of Dinur [Din06] as a way to amplify the “unsatisfiability” of unsatisfiable cnf formulas, which led to a significantly simpler new proof of the famous PCP Theorem. Again, the transparent combinatorial nature of the direct product construction played a major role in getting this simple proof.

In this paper, motivated by the complexity-theoretic applications to hardness amplification, we have studied the direct product construction as a tool for obtaining locally approximately list-decodable error-correcting codes. We improved the previously known decoding algorithms, achieving optimal running time and list size for decoding from a corrupted received word when the amount of corruption is less than $1 - \epsilon$ for a “large” $\epsilon$. As an immediate application, this gives a strong average-case hardness amplification for $\mathsf{P}^{\mathsf{NP}}$ with respect to $\mathsf{BPP}$ algorithms.

The obvious open question is to strengthen the parameters of the decoding algorithm. Ideally, we would like to prove the following dream version of our Main Theorem:

**Dream Version of Direct Product Decoding.** There is a randomized algorithm $A$ that, given a circuit $C$ $\epsilon$-computing the direct product $f^k$ of an $n$-variable Boolean function $f$, outputs with probability at least $\text{poly}(\epsilon)$ a circuit $C'$ that computes $f$ on all but at most $\delta$ fraction of inputs for $\delta \leq O((\ln 1/\epsilon)/k)$. The running time of $A$ is $\text{poly}(|C|/\epsilon)$.

In the present paper, the bottleneck is in our Direct Product amplification procedure. Every step of amplification costs a quadratic decrease in the fraction of tuples where the new circuit (approximately) computes a larger direct product. Is there a more efficient way to do Direct Product amplification?

We hope that our efficient list-decoding algorithm for the direct product codes will also be useful for getting strong hardness amplification within $\mathsf{NP}$. To this end, it would suffice to construct an efficiently approximately list-decodable monotone binary code where every bit of the codeword is a monotone Boolean function of the message, and to concatenate our direct product code with such a monotone code. This direction has been pursued in [BOKS06], where Trevisan’s amplification results for $\mathsf{NP}$ [Tre03, Tre05] are re-proved by combining decoding algorithms for certain monotone codes and the decoding algorithm for the direct-product code from the present paper. Improving Trevisan’s results seems to require efficient list-decoding algorithms for monotone codes.

**Acknowledgements** This paper was inspired by a conversation that the first author had with Luca Trevisan at IAS in the Spring of 2005. We want to thank Luca for his suggestions that were the starting point of this research. We also want to thank Ronen Shaltiel for pointing out to us that the methods in [Raz98] can be used in the proof of our Sampling Lemma (Lemma 10). The third author thanks Josh Buresh-Oppenheim and Rahul Santhanam for interesting discussions. Finally,
we also thank Dieter van Melkebeek for his comments, and Antonina Kolokolova for her help. The first two authors were partially supported by NSF grants 0515332 and 0313241. Views expressed are the authors’ and are not endorsed by the NSF. The third author gratefully acknowledges the support by an NSERC Discovery grant.

References


A Information-theoretic upper bounds

Here we will prove that, both for Yao’s XOR Lemma and for truncated Hadamard codes, the information-theoretic upper bound on the list size for radius $(1/2 - \epsilon)$ is essentially $O(1/\epsilon^2)$, which is the upper bound for Hadamard codes (see, e.g., [GL89]).

First, we will prove the list-size upper bound for the case of Yao’s XOR Lemma; the proof for the case of truncated Hadamard codes will be very similar.

Yao’s XOR Lemma defines the following error-correcting codes. For parameters $n, k \in \mathbb{N}$, a message $msg \in \{0, 1\}^n$ is encoded by a codeword $code_{msg} \in \{0, 1\}^n$, where the positions of $code_{msg}$ are indexed by $k$-tuples $s = (i_1, \ldots, i_k) \in [n]^k$ and $code_{msg}(s) = code_{msg}(i_1, \ldots, i_k) = \oplus_{j=1}^k msg(i_j)$.

Suppose we are given an oracle $B : [n]^k \rightarrow \{0, 1\}$, and are asked to compute the maximum number of messages $msg \in \{0, 1\}^n$ such that $\Pr_{s \in [n]^k}[code_{msg}(s) = B(s)] \geq 1/2 + \epsilon$, for some parameter $\epsilon > 0$. As discussed earlier in the paper, we have to settle for approximate list-decoding, and so the correct question to ask is: What is the maximum value of $t$ such that there exist $t$ $n$-bit strings $m_1, \ldots, m_t$ satisfying the following two conditions:

1. for every two distinct $i, j \in [t]$, $m_i$ and $m_j$ differ in at least $\delta n$ positions, and

2. for each $i \in [t]$, $\Pr_{s \in [n]^k}[code_{m_i}(s) = B(s)] \geq 1/2 + \epsilon$.

The next Theorem shows that if $\delta$ is large, or in other words the messages in the list are not allowed to be close to each other, then the list size can be shown to be small in the parameter $\epsilon$. On the other hand if $\delta$ is small which means that the messages in the list can be close to each other, then the list size is exponentially large.
**Theorem 36.** Let $m_1, \ldots, m_t$ be $t$ strings satisfying the two conditions above. There exist constants $c_1, c_2, c_3$ such that:

1. if $\delta > c_1 \cdot \frac{\log 1/\epsilon}{k}$, then $t = O(1/\epsilon^2)$ and,
2. if $\delta \leq \min \left( c_2 \cdot \frac{\log 1/\epsilon}{k}, c_3 \right)$, then $t = \Omega(2^\delta n)$.

We prove the above theorem by analyzing the two cases separately. We start by analyzing the case when $\delta$ is large.

**List size for large $\delta$** The following lemma gives an upper bound on the list size.

**Lemma 37.** Let $m_1, \ldots, m_t$ be $t$ $n$-bit strings satisfying the following two conditions.

1. for every two distinct $i, j \in [t]$, $m_i$ and $m_j$ differ in at least $\delta n$ positions, and
2. for each $i \in [t]$, $\Pr_{s \in [n]^k}[\text{code}_{m_i}(s) = B(s)] \geq 1/2 + \epsilon$.

Then

$$t \leq \frac{1}{4\epsilon^2 - e^{-2\delta k}}.$$ 

**Proof.** For every $s \in [n]^k$, let

$$\epsilon_s = \Pr_{i \in [t]}[\text{code}_{m_i}(s) = B(s)] - \Pr_{i \in [t]}[\text{code}_{m_i}(s) \neq B(s)] = \frac{1}{t} \sum_{i \in [t]}(-1)^{\text{code}_{m_i}(s) \oplus B(s)}.$$ 

Observe that $\text{Exp}_{s \in [n]^k}[\epsilon_s] \geq 2\epsilon$. So, we get that $4\epsilon^2 \leq (\text{Exp}_{s}[\epsilon_s])^2$. By Jensen’s inequality, the latter is at most $\text{Exp}_{s}[(\epsilon_s)^2]$.

We have

$$\text{Exp}_{s}(\epsilon_s)^2 = \text{Exp}_{s}\left[\frac{1}{t^2} \sum_{i,j} (-1)^{\text{code}_{m_i}(s) \oplus \text{code}_{m_j}(s)}\right]$$

$$= \text{Exp}_{s}\left[\frac{1}{t^2} \sum_{i,j} (-1)^{\text{code}_{m_i \oplus m_j}(s)}\right]$$

$$= \frac{1}{t^2} \text{Exp}_{s}\left[\sum_i (-1)^0 + \sum_{i \neq j} (-1)^{\text{code}_{m_i \oplus m_j}(s)}\right]$$

$$= \frac{1}{t} + \frac{1}{t^2} \sum_{i \neq j} \text{Exp}_{s}[(-1)^{\text{code}_{m_i \oplus m_j}(s)}].$$

Next we bound the quantity $\text{Exp}_{s}[(-1)^{\text{code}_{m_i \oplus m_j}(s)}]$ in the expression above.

**Claim 38.** For any $i \neq j$, we have $\text{Exp}_{s}[(-1)^{\text{code}_{m_i \oplus m_j}(s)}] \leq (1 - 2\delta)^k$.

**Proof.** First, observe that by the assumption on pairwise distance between messages, we have that the string $m' = m_i \oplus m_j$ has relative Hamming weight $w \geq \delta$. We need to compute the probability that a random $k$-tuple $s \in [n]^k$ hits an even number of 1s in the string $m'$ minus the probability that it hits an odd number of 1s. This is exactly

$$\sum_{\text{even } i \in [k]} \binom{k}{i} w^i (1-w)^{k-i} - \sum_{\text{odd } i \in [k]} \binom{k}{i} w^i (1-w)^{k-i} = \sum_{i \in [k]} \binom{k}{i} (-w)^i (1-w)^{k-i}$$

$$= (1-2w)^k.$$
The latter is at most $(1-2\delta)^k$. \hfill \Box

Using the bound from Claim 38, we have
\[
\exp_s(\epsilon_s)^2 \leq \frac{1}{t} + \frac{t(t-1)}{2t^2}(1-2\delta)^k \leq \frac{1}{t} + \frac{1}{2}e^{-2\delta k}.
\]
Recalling that $\exp_s(\epsilon_s)^2 \geq 4\epsilon^2$, we obtain
\[
t \leq \frac{1}{4\epsilon^2 - \frac{1}{2}e^{-2\delta k}},
\]
as required. \hfill \Box

**Remark 39.** The upper bound on the list size in Lemma 37 above is $O(1/\epsilon^2)$ for $\delta > \frac{\ln(1/\epsilon)}{k}$. So, allowing the decoding algorithm to make mistakes on at least $\frac{\ln(1/\epsilon)}{k}$ fraction of positions in the decoded string, makes it possible to have an efficient list decoding algorithm for such XOR based codes. Note that this kind of relationship among $k$, $\delta$, and $\epsilon$ is also assumed in the statement of Yao’s XOR Lemma, and that is why one can hope to have a proof of Yao’s XOR Lemma with the list size $\text{poly}(1/\epsilon)$.

**Lower bound for list size for large $\delta$.** Here, we’ll show that even for large $\delta$, the list size can grow polynomially with $1/\epsilon$. First we argue the case of $k$-truncated Hadamard code (where the encoding of a given $n$-bit message $msg$ is the sequence $\langle msg, r \rangle$ over all $n$-bit strings $r$ of Hamming weight exactly $k$), and then give a simple reduction to the case of $k$-XOR code.

Let $(\binom{n}{k})^{-1/256} < \epsilon < 2^{-n/256}$, and let $\delta < 1/4$. We’ll show that there are $\Omega(1/\epsilon^2)$ strings of length $n$ all of whom are $\delta n$ Hamming distance apart and whose $k$-truncated Hadamard encodings agree with a fixed function $B$ on $1/2 + \epsilon$ fraction of inputs.

Pick $m_1, \ldots, m_{2T^t+1}$ uniformly at random. The probability that any two are within Hamming distance $\delta n$ is at most $O(T^2e^{-(1/4)n/2})$, which is exponentially small as long as $T < 2^{-n/64}$. For $r$ a string of Hamming weight $k$, let $B(r) = \langle \text{maj}_{i \leq 2T^t+1}(m_i, r) \rangle$.

We say that $r$ is balanced for $B$ if the number of indices $i$ with $\langle m_i, r \rangle = 0$ is either $T$ or $T+1$. Note that each fixed $r$ is balanced with probability at least $\Omega(1/\sqrt{T})$, where the probability is over the choice of $m_1, \ldots, m_{2T^t+1}$. Hence the expected fraction of balanced $r$’s is at least $\Omega(1/\sqrt{T})$. By averaging, with probability at least $\Omega(1/\sqrt{T})$ over the choice of $m_1, \ldots, m_{2T^t+1}$, there are at least $\Omega(1/\sqrt{T})$ fraction of balanced $r$’s.

Fix any index $i$, $1 \leq i \leq t$, and fix all $m_j$’s for $j \neq i$. We say that $r$ is $i$-balanced if the number of $j \neq i$ with $\langle m_j, r \rangle = 0$ is $T$. Note that, for $r$’s that are $i$-balanced, $B(r) = \langle m_i, r \rangle$ whatever $m_i$ is. Also, note that, for $r$ not $i$-balanced, $B(r)$ is determined, and for such $r$’s, the bits $\langle m_i, r \rangle$’s are uniform and pairwise independent.

For a fixed $i$ and fixed $m_j$’s for $j \neq i$, suppose that the fraction of $i$-balanced $r$’s is at least $\Omega(1/\sqrt{T})$. Then, as just observed, $\langle m_i, r \rangle = B(r)$ for at least $\Omega(1/\sqrt{T})$ fraction of $i$-balanced $r$’s, independent of the choice of $m_i$. For the remaining $r$’s, define random variables $X_r$ where $X_r = 1$ if $\langle m_i, r \rangle = B(r)$, and $X_r = 0$ otherwise. As we observed above, these random variables are pairwise independent and individually uniform (over the random choice of $m_i$). By Chebyshev, the probability that there are fewer than $1/2 - O(1/\sqrt{T})$ fractions of $r$’s with $X_r = 1$ is less than $O(T/\binom{n}{k})$. Observe that if this latter event does not happen, then we have that the the correlation between $\langle m_i, r \rangle$ and $B(r)$ is at least $\Omega(1/\sqrt{T})$.

34
Thus, the probability that there is at least one $i$ with $\Omega(1/\sqrt{T})$ $i$-balanced $r$’s, such that for this $i$ the correlation between $\langle m_i, r \rangle$ and $B(r)$ is not $\Omega(1/\sqrt{T})$, is at most $O(T^2/(n^4))$. As long as $T = o((\log n)^{1/3})$, with high probability all of the $i$’s with $\Omega(1/\sqrt{T})$ $i$-balanced $r$’s have correlation $\Omega(1/\sqrt{T})$ with $B$, even conditioned on the event that there are $\Omega(1/\sqrt{T})$ balanced $r$’s.

On the other hand, each $r$ that is balanced is $i$-balanced for $T + 1$ values of $i$. So if there are $\Omega(1/\sqrt{T})$ balanced $r$’s, then there are $\Omega(T)$ $i$’s with $\Omega(1/\sqrt{T})$ $i$-balanced $r$’s.

Therefore, there must exist a choice of $m$’s and a subset of $\Omega(T)$ of the $i$’s so that each $m_i$ has a correlation of $\epsilon = \Omega(1/\sqrt{T})$ with $B$. Setting $T = 1/\epsilon^2$ concludes the argument for the case of $k$-truncated Hadamard code.

Now we reduce the case of $k$-XOR code to the case of $k$-truncated Hadamard code. Recall that the $k$-XOR encoding of an $n$-bit message $m$ is the sequence of $m(i_1) \oplus \cdots \oplus m(i_k)$ over all $k$-tuples of indices $i_1, \ldots, i_k$ from $[n]$. The fraction of those $k$-tuples $i_1, \ldots, i_k$ that contain some index $j \in [n]$ more than once is at most $k^2/n$, which can be made very small by making $n$ much larger than $k$.

Ignoring the $k$-tuples with repeats, we can partition the remaining $k$-tuples into $\ell$ blocks where each block contains $\binom{n}{k}$ tuples corresponding to distinct $k$-size subsets of $[n]$. For each such block, the $k$-XOR encoding of a given message $m$ (restricted to the $k$-tuples in the block) coincides with the $k$-truncated Hadamard encoding of $m$. So, the $k$-XOR encoding of $m$ restricted to the $k$-tuples without repeats is just a concatenation of $\ell$ copies of the $k$-truncated Hadamard encoding of $m$.

By what we have shown above, there is a collection of $\Omega(1/\epsilon^2)$ $n$-bit messages (pairwise Hamming distance $\delta n$ apart) and a string $B$ such that the $k$-truncated encoding of each message agrees with $B$ in at least $1/2 + \Omega(\epsilon)$ fraction of positions. Let $B'$ be the string obtained as a concatenation of $\ell$ copies of the string $B$. It follows that for the same collection of messages, their $k$-XOR encodings will agree with the string $B'$ in at least $1/2 + \Omega(\epsilon)$ fraction of positions, when the positions are restricted to the $k$-tuples without repeats. Let us now pad $B'$ with enough $0$’s to get the string of length $n^k$. Let us call the new string $B''$. We have that the $k$-XOR encodings of our messages will agree with $B''$ in at least $1/2 + \Omega(\epsilon) - k^2/n$ fraction of positions. The latter can be made $1/2 + \Omega(\epsilon)$, by choosing $n$ large enough so that $k^2/n \ll \epsilon$.

**List size for small $\delta$.** We showed that the list size is small when $\delta > \frac{\ln(1/\epsilon)}{k}$. Next, we show that the list size could be exponentially large when we allow the list to contain messages which are not far apart from one another.

Consider a string $C = \{0\}^{n^k}$ and a list of messages such that all the messages in the list have Hamming weight exactly $\delta n$. We will first show that the codewords for all these messages agree with $C$ on at least $(1/2 + \epsilon)$ fraction of positions. We then show that there is a large subset of messages in the list which are not too close to one another.

**Lemma 40.** Let $m_1, \ldots, m_t$ be $n$-bit strings having hamming weight $\delta n$ and let $\delta \leq \min\left(\frac{\ln 1/\epsilon}{4k}, 1/3\right)$. Then

$$\forall i, \Pr_{s \in [n]^k}[\text{code}_{m_i}(s) = C(s)] = 1/2 + 1/2(1 - 2\delta)^k \geq 1/2 + \epsilon.$$  

**Proof.** For a given message $m_i$, $\text{code}_{m_i}(s) = C(s)$ when $s$ has an even intersection with the subset of positions in $m_i$ which are $1$. The probability of this event is exactly $\sum_{i \in [k]} \binom{k}{i} \delta^i (1 - \delta)^{k - i}$ which is $1/2 + 1/2(1 - 2\delta)^k \geq 1/2 + \epsilon$ when $\delta \leq \min\left(\frac{\ln 1/\epsilon}{4k}, 1/3\right)$. \hfill $\square$

**Lemma 41.** Let $M = \{m_1, \ldots, m_t\}$ be a set of $n$-bit strings having Hamming weight $\delta n$, where $\delta \leq 1/17$. Then there is a subset $N \subseteq M$ such that for any two messages in $N$ the messages differ in at least $\delta n$ positions and $|N| \geq 2^{\delta n - 1}$.

35
Proof. Consider a graph with the messages in $M$ denoting the vertices and there is an edge between two messages if they differ in less than $\delta n$ positions. Given this, an independent set in this graph is a possible candidate for $N$. We pick a maximal independent set $I$ in the graph by the process of picking a vertex, deleting its neighboring vertices and repeating. Set $N = I$. The size of $N$ is lower bounded by $|M|/(\Delta + 1)$, where $\Delta$ is the maximum degree of the graph. We have

$$\Delta = \sum_{i=1}^{\delta n/2} \binom{\delta n}{i} \binom{(1-\delta)n}{i} \leq \binom{(1-\delta)n}{\delta n/2} \sum_{i=1}^{\delta n/2} \binom{\delta n}{i}$$

(since $\delta \leq 1/17$)

$$\leq \binom{(1-\delta)n}{\delta n/2} \cdot 2^{\delta n} \leq 2^{\delta n} \cdot \binom{n}{\delta n/2}.$$  

This yields

$$|N| \geq \frac{n}{\Delta + 1} \geq \frac{n}{2 \cdot 2^{\delta n} \cdot \binom{n}{\delta n/2}} \geq \frac{1}{2^{\delta n+1}} \cdot \frac{n(n-1) \ldots (n-\delta n+1)}{n(n-1) \ldots (n-\delta n/2+1)} \cdot \frac{(\delta n)!}{(\delta n/2)!}$$

$$= \frac{1}{2^{\delta n+1}} \cdot \frac{(n-\delta n/2)(n-\delta n/2-1) \ldots (n-\delta n+1)}{(\delta n)(\delta n-1) \ldots (\delta n/2+1)}$$

$$\geq \frac{(1/\delta - 1)\delta n/2}{2^{\delta n+1}}$$

$$\geq 2^{\delta n-1} \quad (for \quad \delta \leq 1/17).$$

Proof of Theorem 36. The proof easily follows from Lemmas 37, 40, and 41.

B Proof of Lemma 19

For convenience we re-state the lemma.

Lemma 19. Let $S$ be the collection of all $m$-size sets $s \in [k]$, for any $m \leq k$. Let $G \subseteq S$ be any subset of $S$ that has weight at least $\epsilon$ under the uniform distribution over $S$. Let $U$ be the uniform distribution on elements $j \in [k]$, and let $D$ be the distribution defined as follows: pick a set $s \in G$ uniformly at random and output a random element of $s$. Then $\text{Dist}(D, U) \leq O\sqrt{\frac{\log(m/\epsilon)}{m}}$.

We will need the following property of integer-valued random variables.

Lemma 42. Let $X$ be any integer-valued random variable taking values in the set $\{0, 1, \ldots, m\}$. Let $\mu = E[X]$ be the expectation of $X$. Then $\Pr[X \geq \lfloor \mu \rfloor] \geq 1/m$ and $\Pr[X \leq \lceil \mu \rceil] \geq 1/(m+1)$. 

36
Proof. By definition, $\mu = \sum_{i=0}^{m} i \Pr[X = i]$. We can split this sum into two sums as follows:

$$
\mu = \sum_{0<i<\lceil \mu \rceil} i \Pr[X = i] + \sum_{\lceil \mu \rceil \leq i \leq m} i \Pr[X = i].
$$

Using the fact that $X$ is integer-valued, we can bound these two sums by

$$(\mu - 1) \sum_{0<i<\lceil \mu \rceil} \Pr[X = i] + m \sum_{\lceil \mu \rceil \leq i \leq m} \Pr[X = i] \leq (\mu - 1) + m \Pr[X \geq \lceil \mu \rceil].$$

This implies that $\Pr[X \geq \lceil \mu \rceil] \geq 1/m$, as claimed in the first inequality of the lemma.

To prove the second inequality, we observe that $\Pr[X > \lceil \mu \rceil] = \Pr[X \geq \lceil \mu \rceil + 1] \leq \frac{\mu}{\lceil \mu \rceil + 1}$, where we used the fact that $X$ is integer-valued and then applied Markov’s inequality. Finally, we have

$$\Pr[X \leq \lceil \mu \rceil] \geq 1 - \frac{\mu}{\lceil \mu \rceil + 1} = \frac{\lceil \mu \rceil + 1 - \mu}{\lceil \mu \rceil + 1} \geq \frac{1}{m+1}. \quad \square$$

Now we can prove Lemma 19.

Proof of Lemma 19. Suppose that the statistical distance between $U$ and $D$ is $\delta$. It means that there is a statistical test $T \subseteq [k]$ such that the probabilities assigned to $T$ by $U$ and $D$ differ by $\delta$. We shall assume that $\Pr_D[T] \geq \Pr_U[T] + \delta$; the case where $\Pr_U[T] \geq \Pr_D[T] + \delta$ is proved similarly.

We will view each set $x = \{i_1, \ldots, i_m\}$ of size $m$ as an increasing sequence of $m$ elements $i_1 < \cdots < i_m$. Uniformly sampling a set from $G$ will be denoted by $(x_1, \ldots, x_m) \leftarrow G$. Also, for an event $E$, we use $\chi[E]$ as the 0-1 indicator variable of $E$. We have

$$
\mu \overset{def}{=} \mathbb{E}_{\bar{x}=(x_1, \ldots, x_m) \leftarrow G} \left[ \sum_{i=1}^{m} \chi[x_i \in T] \right] = \sum_{i=1}^{m} \mathbb{E}_{\bar{x} \leftarrow G} [\chi[x_i \in T]]
= m \mathbb{E}_{i \leftarrow [m], \bar{x} \leftarrow G} [\chi[x_i \in T]]
= m \Pr_D[T]
\geq m (\Pr_U[T] + \delta)
= m(\lceil |T|/k \rceil + \delta).
$$

On the other hand, by Lemma 42, we get $\Pr_{\bar{x} \leftarrow G} [\sum_{i=1}^{m} \chi[x_i \in T] \geq \mu - 1] \geq 1/m$. Since $G$ has weight at least $\epsilon$ in the collection $S$ of all size-$m$ subsets of $[k]$, we have

$$
\Pr_{\bar{x} \leftarrow S} [\sum_{i=1}^{m} \chi[x_i \in T] \geq \mu - 1] \geq \epsilon/m. \quad (13)
$$

The expectation of $\sum_{i=1}^{m} \chi[x_i \in T]$ under the uniform distribution of $\bar{x}$ in $S$ is $m|T|/k$. Applying the Hoeffding bound to the left-hand side of Equation (13), we get

$$
\Pr_{\bar{x} \leftarrow S} [\sum_{i=1}^{m} \chi[x_i \in T] \geq m|T|/k + m\delta - 1] \leq e^{-\Omega(m\delta^2)}.
$$

Combining this with Equation (13), we conclude that $\Omega(m\delta^2) \leq \log(m/\epsilon)$, and so $\delta \leq O(\sqrt{\log(m/\epsilon)}/m)$, as required. \qed