Electronic Colloquium on Computational Complexity, Report No. 105 (2023)

Derandomization vs Refutation: A Unified Framework for Characterizing Derandomization

Lijie Chen [*] Roei Tell [†] Ryan Williams [‡]

July 13, 2023

Abstract

We establish an equivalence between two algorithmic tasks: **derandomization**, the deterministic simulation of probabilistic algorithms; and **refutation**, the deterministic construction of inputs on which a given probabilistic algorithm fails to compute a certain hard function.

We prove that refuting low-space probabilistic streaming algorithms that try to compute functions $f \in \mathcal{FP}$ is equivalent to proving that $pr\mathcal{BPP} = pr\mathcal{P}$, even in cases where a lower bound for f against this class (without a refuter) is already unconditionally known. We also demonstrate the generality of the connection between refutation and derandomization, by establishing connections between refuting classes of constant-depth circuits of sublinear size and derandomizing constant-depth circuits of polynomial size with threshold gates (i.e., $\mathcal{TC}^0$).

Our connection generalizes and strengthens recent work on the characterization of derandomization. In particular, using refuter terminology allows to directly compare several recent works to each other and to the current work, as well as to chart a path for further progress. Along the way, we also improve the targeted hitting-set generator of Chen and Tell (FOCS 2021), showing that its translation of hardness to pseudorandomness scales down to $\mathcal{TC}^0$.

[*]Miller Institute for Basic Research in Science at University of California, Berkeley, CA. Email: wjmzbmr@gmail.com

[†]The Institute for Advanced Study at Princeton NJ and the DIMACS Center at Rutgers University, NJ. Supported in part by the NSF under grant numbers CCF-1445755 and CCF-1900460. Email: roeitell@gmail.com

[‡]MIT, Cambridge, MA. Supported in part by the Simons Institute at UC Berkeley, NSF CCF-2127597, and a Frank Quick Faculty Research Innovation Fellowship. Email: rrw@mit.edu

ISSN 1433-8092

Contents

1 Introduction **1**
- 1.1 Derandomization of $pr\mathcal{BPP}$ vs refutation for low-space streaming algorithms . . . 3
- 1.2 Scaling down the equivalence to weak circuit classes . . . 4
- 1.3 Generalizing previous characterizations of derandomization . . . 6
- 1.4 Refuters against deterministic algorithms and the Lossy Code problem . . . 8

2 Technical overview **9**
- 2.1 Our starting point: A new perspective . . . 10
- 2.2 Warm-up: The Nisan-Wigderson generator . . . 10
- 2.3 A broader class of hard functions . . . 11
- 2.4 Extending the connection down to $\mathcal{TC}^0$, and an improved Chen-Tell generator . . . 12

3 Preliminaries **16**

4 A $\mathcal{TC}^0$-locally-encodable and $\mathcal{TC}^0$-locally-approximately-decodable code **24**
- 4.1 The first code: From distance $N^{-\Omega(1)}$ to distance 0.01 . . . 24
- 4.2 The second code: From distance 0.01 to distance $1/2 - N^{-\Omega(1)}$ . . . 30
- 4.3 Proof of Proposition 4.1 . . . 34

5 Improved Chen-Tell hitting set generator with $\mathcal{TC}^0$ reconstruction **37**
- 5.1 Efficient polynomial decompositions of highly uniform threshold circuits . . . 37
- 5.2 Reconstructive targeted HSG for highly uniform $\mathcal{TC}^0$ circuits . . . 43

6 Derandomization vs refutation **48**
- 6.1 Derandomization vs refutation against low-space streaming algorithms . . . 48
- 6.2 Derandomization vs refutation for $\mathcal{TC}^0$ . . . 52
- 6.3 Refuting deterministic streaming algorithms vs Lossy Code . . . 57

7 Characterization of derandomization via the refuter framework **59**
- 7.1 Leakage-resilient hardness and refuter for Identity . . . 59
- 7.2 Hardness of Conditional Kolmogorov Complexity . . . 61
- 7.3 Almost-all-inputs hardness . . . 62

A The tarHSG of [CT21] with low-space streaming reconstruction **67**

B The STV PRG with $\mathcal{TC}^0 \circ \mathsf{XOR}$ reconstruction **68**

1 Introduction

Can every randomized algorithm be simulated by a deterministic one, with low overhead? The question of whether universal derandomization is possible, generally expressed as $pr\mathcal{BPP} = pr\mathcal{P}$, has fascinated a generation of researchers, partly due to deep connections between derandomization and computational lower bounds. In the classical "hardness vs randomness" line of works, efficient derandomization (e.g., $pr\mathcal{BPP} = pr\mathcal{P}$) was shown to be possible, assuming exponentially-strong non-uniform circuit lower bounds against exponential time (see, e.g., [NW94, IW97, STV01, SU05, Uma03]). That is, it has been known for a long time that sufficiently strong non-uniform circuit lower bounds would imply universal derandomization.

However, non-uniform circuit lower bound hypotheses appear to be overkill for $pr\mathcal{BPP} = pr\mathcal{P}$, since $pr\mathcal{BPP} = pr\mathcal{P}$ is only concerned with derandomizing probabilistic uniform algorithms (e.g., Turing machines). More recently, researchers have found potentially weaker uniform lower bound assumptions which suffice (and are sometimes *equivalent*) for $pr\mathcal{BPP} = pr\mathcal{P}$:

- Chen and Tell [CT21] show that $pr\mathcal{BPP} = pr\mathcal{P}$ follows from the assumption that there is a multi-output function f computable by poly-size LOGSPACE-uniform circuits of depth n^2 that cannot be computed on almost all inputs[1] by any probabilistic *fixed*-polynomial-time algorithm (running faster than the deterministic poly-time algorithm for f). They also prove that the assumption is necessary when the depth restriction is removed.

- Liu and Pass [LP22a] show that $pr\mathcal{BPP} = pr\mathcal{P}$ is equivalent to proving a certain lower bound on probabilistic polynomial-time algorithms attempting to approximate the conditional Kt (Levin) complexity of a given binary string. In follow-up work [LP22b], they show that $pr\mathcal{BPP} = pr\mathcal{P}$ is equivalent to the existence of a poly-time f which is "leakage-resilient" against probabilistic fixed-polynomial-time algorithms on almost all inputs.

- Korten [Kor22a] showed that $pr\mathcal{BPP} = pr\mathcal{P}$ is equivalent to constructing a deterministic polynomial-time algorithm that gets as input a probabilistic circuit $C\colon \{0,1\}^n \to \{0,1\}^{n-1}$ and a deterministic circuit $D\colon \{0,1\}^{n-1} \to \{0,1\}^n$, and outputs $x \in \{0,1\}^n$ such that $\Pr[D(C(x)) = x] < 1/2$.

In a different setting, a recent related work of van Melkebeek and Sdroievski [vMS23] shows similar results for proving that $\mathcal{AM} = \mathcal{NP}$.

It is not *a priori* clear how to directly compare the various assumptions in the above works, all of which were proved to be equivalent to universal derandomization.

Efficient refutations. Another line of works, dating back to [Kab01] (see also [GSTS07]), studies *efficient refutation*. Suppose we know a lower bound "$f \notin \mathcal{C}$" for some class of algorithms $\mathcal{C}$. The problem of efficient refutation asks how easy it is to produce "bad" inputs, on which a given "weak" algorithm $A \in \mathcal{C}$ fails to compute f. More formally, for a class $\mathcal{C}$ of algorithms (circuits, Turing machines, streaming algorithms, etc.) and a function $f\colon \{0,1\}^* \to \{0,1\}^*$, we say an algorithm A is a **refuter for f against $\mathcal{C}$** if for "many" n and all $C \in \mathcal{C}$, $A(1^n, \langle C\rangle)$ outputs

[1]Throughout the paper, the meaning of "almost all inputs" will be "all but finitely many inputs"; that is, every probabilistic machine succeeds in computing the function only on finitely many inputs.

$x \in \{0,1\}^n$ such that $C(x) \neq f(x)$.[2] A lower bound of the form "$f \notin \mathcal{C}$" is said to be **constructive** if there is a efficient refuter for f against $\mathcal{C}$, e.g. there is a refuter computable in polynomial time.

A recent work by Chen, Jin, Santhanam, and Williams [CJSW21] showed that for a variety of *unconditionally known* lower bounds, constructivizing these bounds (that is, finding efficient refuters for them) would have significant consequences in complexity theory. Most pertinently to the current work, they showed that *sufficiently strong refutation implies derandomization*: if there exist polynomial-time refuters against *nondeterministic* models (one-tape Turing machines, as well as streaming algorithms), then E needs exponential-size circuits, which in turn implies $pr\mathcal{BPP} = pr\mathcal{P}$.[3] Indeed, their results use the classical approach for derandomization, which relies on strong circuit lower bounds, rather than the new approaches, which use uniform lower bounds.

Our contributions: Bird's eye view. The main question motivating this work is whether we can leverage the new approach for derandomization in order to prove stronger connections between refutation and derandomization. For example, can we show that more relaxed forms of refutation (compared to the ones studied in [CJSW21]) suffice for derandomization? Taking this question even further: Can we show that refutation is *equivalent* to derandomization, connecting the study of refuters to the line of works proving *characterizations* of the $pr\mathcal{BPP} = pr\mathcal{P}$ conjecture?

We provide a strong affirmative answer to the foregoing questions, by proving a general equivalence between derandomization and refutation. In fact, our refuter-based characterization of derandomization generalizes and significantly strengthens all the recently discovered results studying derandomization from weaker hypotheses (i.e., [CT21, LP22a, LP22b, Kor22a]). It turns out that looking at derandomization through the lens of refutation allows us to directly compare the hypotheses in each of these works, as well as to prove stronger results.

In more detail, we study the consequences of *deterministically refuting* classes of *probabilistic algorithms*, for hard functions in $\mathcal{FP}$. We show that this sort of refutation – even for *unconditionally known* lower bounds – is equivalent to derandomization. Moreover, we prove that this equivalence holds both for general probabilistic algorithms and for weak classes of algorithms: the equivalence (or near-equivalence) scales down "as far as" $\mathcal{TC}^0$, which is a lower complexity class compared to previous works studying derandomization from weaker hypotheses.

Setup and notation. We consider refuting non-uniform classes $\mathcal{C}$ of algorithms: for every input length n, $\mathcal{C}$ contains a set $\mathcal{C}_n$ of probabilistic algorithms. The algorithms in $\mathcal{C}_n$ do not need to be Boolean circuits, as in the usual definition of non-uniform classes; for example, $\mathcal{C}_n$ could be a set of probabilistic RAM machines or streaming algorithms with a certain description length and runtime bound, where we consider their execution on inputs of fixed length n.

We say that A is a **refuter for a function** f **against a class** $\mathcal{C} = \cup_{n\in\mathbb{N}}\mathcal{C}_n$ of probabilistic algorithms if for every $n \in \mathbb{N}$ and every $C \in \mathcal{C}_n$, $A(1^n, \langle C\rangle)$ outputs an $x \in \{0,1\}^n$ such that $\Pr[C(x) = f(x)] < 2/3$, where the probability is over the internal randomness of C. If A runs in deterministic polynomial time, we say that A is an $\mathcal{FP}$**-refuter**. We say that A is a $\mathcal{BPP}$**-refuter for** f **against** $\mathcal{C}$ if A runs in probabilistic polynomial time and satisfies

$$\Pr\Big[A(1^n, \langle C\rangle) \text{ outputs an } x \in \{0,1\}^n \text{ such that } \Pr[C(x) = f(x)] < 2/3\Big] \geq 2/3$$

[2] The "many" n may be infinitely many n, or all but finitely many n, depending on the lower bound being proved.

[3] They also showed that any proof of classical conjectured lower bounds (such as $\mathcal{NEXP} \neq \mathcal{BPP}$) would necessarily yield constructive lower bounds; that is, constructivity is necessary for proving these conjectures.

for every $n \in \mathbb{N}$ and every $C \in \mathcal{C}_n$, where the outer probability is over the randomness of A.

1.1 Derandomization of $pr\mathcal{BPP}$ vs refutation for low-space streaming algorithms

Define str-$\mathcal{TISP}[t(n), s(n)]$ as the class of probabilistic one-pass streaming algorithms that on n-bit inputs have description length n, and run in time $t(n)$ and space $s(n)$. Our first result asserts that constructing an $\mathcal{FP}$-refuter for any function in $f \in \mathcal{FP}$ against *low-space streaming algorithms* suffices for derandomization. (This should be compared with [CJSW21, Theorems 1.5 and 3.4], which needed refuters for *general non-deterministic machines*.) In fact, we prove an *equivalence* between such refutation and $pr\mathcal{BPP} = pr\mathcal{P}$, as follows:

Theorem 1.1. *The following statements are equivalent:*

1. *For some $\varepsilon > 0$ and $f\colon \{0,1\}^* \to \{0,1\}^*$ computable in polynomial time T, there is an $\mathcal{FP}$-refuter for f against* str-$\mathcal{TISP}[T(n)^{1+\varepsilon}, n^{\varepsilon}]$.
2. *$pr\mathcal{BPP} = pr\mathcal{P}$.*
3. *For every class $\mathcal{C}$ of probabilistic RAMs supporting error-reduction[4], and every $f \in \mathcal{FP}$ such that there is a $\mathcal{BPP}$-refuter for f against $\mathcal{C}$, there is an $\mathcal{FP}$-refuter for f against $\mathcal{C}$.*

Theorem 1.1 states multiple compelling equivalences. First of all, it says that universal derandomization is equivalent to derandomizing refuters against *efficient low-space streaming algorithms*. We find this equivalence particularly surprising, since this class of algorithms seems remarkably weak. We also stress that there are many known *unconditional lower bounds* for functions in polynomial time against streaming algorithms with space $o(n)$ and *any* running time (see, e.g., [AMS99]). Thus, one implication of Theorem 1.1 is that *constructivizing known lower bounds for streaming algorithms suffices to prove that $pr\mathcal{BPP} = pr\mathcal{P}$*.

Second, Theorem 1.1 also states that universal derandomization ($pr\mathcal{BPP} = pr\mathcal{P}$) is equivalent to derandomizing every probabilistic polynomial-time refuter against a class of probabilistic RAMs: when a probabilistic efficient refuter exists, there is also a deterministic one. Therefore, derandomizing probabilistic refuters is "complete" for universal derandomization.

Third, Theorem 1.1 says that deterministically refuting *streaming algorithms* is equivalent to deterministically refuting *significantly stronger classes $\mathcal{C}$*. For example, constructivizing lower bounds for certain functions in quasilinear time against $n^{2-\varepsilon}$-time and n^{ε}-space *streaming algorithms* (e.g., constructivizing lower bounds in [AMS99]) would also constructivize lower bounds for certain multi-output functions in quasilinear time against *general $n^{2-\varepsilon}$-time and n^{ε}-space algorithms* (e.g., it would constructivize lower bounds such as those in [Bea91, MW18]).[5]

Refuters for functions with multiple output bits. The reader might have noticed that the function f in Theorem 1.1 is allowed to have multiple output bits. This generalization is important: constructing refuters for functions with multiple output bits is, intuitively, a significantly easier task than constructing refuters for decision problems. Thus, our results offer a characterization

[4]Informally, we only require that in $\mathcal{C}$, we can take the majority vote of constantly many independent runs of an algorithm in $\mathcal{C}$; see Definition 6.3 for details.

[5]This can be viewed as a generalization and strengthening of [LP22b, Theorem 1.2], who showed that leakage-resilient hardness with n^{ε} bits of leakage is equivalent to leakage-resilient hardness with $n - O(\log(n))$ bits of leakage, by proving that both are equivalent to derandomization (where hardness here is in the "almost all inputs" sense).

of derandomization in terms of weaker hypotheses. Moreover, it is through the use of multiple output bits that we are able to generalize and strengthen the known characterizations of derandomization from [CT21, LP22a, LP22b, Kor22a], as well as compare them to each other (we elaborate on this in Section 1.3).

In contrast to the proofs of our results (some of which are quite involved), it is easy to show that refuters for functions with a single output bit implies derandomization (see Section 3.5), and indeed the latter statement has fewer interesting consequences.[6]

1.2 Scaling down the equivalence to weak circuit classes

We demonstrate the generality of the connection between refutation and derandomization by showing that the equivalence in Theorem 1.1 scales down to weak complexity classes. In fact, we show that this equivalence scales "as far down" as $\mathcal{TC}^0$, which is a lower complexity class than in [CT21, LP22a, LP22b]. As this scaling-down requires significant technical work, we will only illustrate this for the "extreme point" of $\mathcal{TC}^0$; we have no reason to doubt that similar equivalences hold for stronger classes such as $\mathcal{NC}$. A secondary reason for proving scaled-down equivalences is a hope that our results could be leveraged in order to prove unconditional derandomizations for weaker circuit classes.

In the following, we show connections between refuting classes of probabilistic circuits with constant depth and a *sub-linear number of gates*, and derandomization of constant-depth circuit families of polynomial size with threshold gates, a.k.a. $\mathcal{TC}^0$.[7] Towards stating the results, recall that CAPP is the problem in which we are given a circuit $C\colon \{0,1\}^n \to \{0,1\}$ and want to distinguish between the case $\Pr_r[C(r) = 1] \geq 2/3$ and $\Pr_r[C(r) = 1] \leq 1/3$. This problem is $pr\mathcal{BPP}$-complete, in that CAPP is solvable in deterministic polynomial time if and only if $pr\mathcal{BPP} = pr\mathcal{P}$. Also recall that in $\mathsf{CAPP}_{0,1/2}$, we are given a circuit $C\colon \{0,1\}^n \to \{0,1\}$ and have to distinguish between the cases $\Pr_r[C(r) = 1] \geq 1/2$ and $\Pr_r[C(r) = 1] = 0$. This "one-sided" CAPP problem is solvable in deterministic polynomial time if and only if $pr\mathcal{RP} = pr\mathcal{P}$.

Full equivalence for a specific function. We first consider refuters only for the specific "hard" function $f(x) = x$, denoted $\mathsf{Identity}$. Indeed, extremely weak algorithms fail to compute $\mathsf{Identity}$ (e.g., algorithms that only access n^{ε} bits of input), and we show that refuters for $\mathsf{Identity}$ against certain such classes is equivalent to solving CAPP in polynomial time, for all of $\mathcal{TC}^0$.

Theorem 1.2. *The following are equivalent:*

1. *There is a polynomial-time algorithm solving* CAPP *for* $\mathcal{TC}^0$ *circuits.*
2. *For some* $\varepsilon > 0$*, there is an* $\mathcal{FP}$*-refuter for* $\mathsf{Identity}$ *against probabilistic* $\mathcal{TC}^0 \circ \oplus$ *circuits that have* $O(n^{1+\varepsilon})$ *wires, and* n^{ε} *gates in the bottom* XOR *layer.*

[6] This situation is reminiscent of that in Chen and Tell [CT21]. To see this, note that $pr\mathcal{BPP} = pr\mathcal{P}$ trivially follows from the existence of $f \in \mathcal{P}$ such that for all but finitely many inputs x, $\Pr_r[M(x,r) = f(x)] < 2/3$, where M is a probabilistic machine solving the $pr\mathcal{BPP}$-complete decision problem CAPP. The main contribution of [CT21] is proving that $pr\mathcal{BPP} = pr\mathcal{P}$ follows from a similar statement for functions with *multiple* output bits. (The original statement in [CT21] asserts that $pr\mathcal{BPP} = pr\mathcal{P}$ follows from the existence of f that is hard for *all* probabilistic machines running in some fixed polynomial time; but since it suffices to derandomize a machine solving a $pr\mathcal{BPP}$-complete problem, it suffices to require that f will be hard on almost all inputs for a single (specific) machine F.)

[7] Throughout the paper, we restrict the gates in $\mathcal{TC}^0$ circuits to have *polynomially bounded* weights; see Section 3.1.

As in Theorem 1.1, the refuted class in Theorem 1.2 is very weak. In particular, for $\varepsilon < 1$ we already unconditionally know that Identity cannot be computed by $\mathcal{TC}^0 \circ \oplus$ circuits as in Theorem 1.2; what we lack is an $\mathcal{FP}$-refuter "witnessing" the simple lower bound.[8]

Near-equivalence for a broader class of hard functions. Theorem 1.2 shows a full equivalence, but needs a refuter for the specific function Identity. We now relax the hypothesis by allowing refuters for a significantly richer class of hard functions, at the cost of proving a near-equivalence rather than a full equivalence. Details follow.

For a $\mathcal{TC}^0$ circuit C with $T(n)$ gates, consider the function $\Phi(i,j) = w_{i,j}$, where $i \in [T(n)]$ is the index of a threshold gate g of C, $j \in [T(n)]$ is the index j of a child h of g in C, and $w_{i,j}$ is the weight of h in the linear combination defining g. Roughly speaking, we say that a circuit C is **highly uniform** if Φ is computable by $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuits of size $T^{o(1)}$ (see Definition 3.6).

Consider any f computable by highly uniform $\mathcal{TC}^0$ circuits. In one direction (refutation $\Rightarrow$ derandomization), we show that a refuter for f against distributions over $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits of *n^ε gates*, where $\varepsilon \in (0,1)$ is an arbitrarily small constant, would suffice to solve $\mathsf{CAPP}_{0,1/2}$ for all of $\mathcal{TC}^0$. (As usual, the notation SUM denotes gates that compute a weighted sum of their inputs with polynomially bounded weights, over the integers; see Section 3.2.1.)

Theorem 1.3 (informal, see Theorem 6.10)**.** *For every $\varepsilon > 0$ and $d,k \in \mathbb{N}$ there exists $d' > 1$ such that the following holds. Let $f\colon \{0,1\}^\star \to \{0,1\}^\star$ be any function mapping n bits to n^ε bits that is computable by a family of highly uniform threshold circuits of depth d and size n^k. Assume that there is a $\mathcal{P}$-computable refuter for f against distributions over $\mathcal{TC}^0_{d'} \circ \mathsf{SUM}$ circuits with $n^{2\varepsilon}$ gates. Then, there is a deterministic polynomial-time algorithm solving $\mathsf{CAPP}_{0,1/2}$ for $\mathcal{TC}^0_d$ circuits.*

Similarly to our previous results, hard functions as in Theorem 1.3 exist, for example the inner product mod 2 (IP2) function.[9] The challenge is in constructivizing the lower bound.

To complement Theorem 1.3 and show a near-equivalence, we will slightly restrict the class of hard functions and the class of refuted algorithms. For a family of distributions $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ where $\mathcal{D}_n$ is over n-bit $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits, we say that $\mathcal{D}$ is $\mathcal{TC}^0$**-samplable** if for every $n \in \mathbb{N}$ there exists a multi-output $\mathcal{TC}^0$ circuit S_n, called a *sampler*, such that the output distribution of S_n over random input is $\mathcal{D}_n$ (see Definition 3.8 for details). Then:

Theorem 1.4 (informal, see Theorem 6.13)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}^{n^\varepsilon}$ be computable by highly uniform $\mathcal{TC}^0$ circuits, and assume that there is a probabilistic $\mathcal{TC}^0$-computable refuter for f against $\mathsf{Samp}\text{-}\mathcal{TC}^0[n^{2\varepsilon}]$, where $\mathsf{Samp}\text{-}\mathcal{TC}^0[n^{2\varepsilon}]$ is the class of $\mathcal{TC}^0$-samplable distributions over $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits with $n^{2\varepsilon}$ gates. Then, for the following three statements, we have $(1) \implies (2) \implies (3)$.*

1. *There is a deterministic polynomial-time algorithm solving* CAPP *for $\mathcal{TC}^0$.*
2. *There is an $\mathcal{FP}$-refuter for f against $\mathsf{Samp}\text{-}\mathcal{TC}^0[n^{2\varepsilon}]$.*
3. *There is a deterministic polynomial-time algorithm solving $\mathsf{CAPP}_{0,1/2}$ for $\mathcal{TC}^0$.*

[8] Since there are only n^ε XOR gates in the bottom layer, all functions computed by probabilistic $\mathcal{TC}^0 \circ \oplus$ circuits have two-party (public-coin) probabilistic communication complexity $O(n^\varepsilon)$. For all $\varepsilon < 1$, such protocols cannot compute Identity, as this would require both parties to completely reconstruct the opposite party's $n/2$-bit input.

[9] Following [Nis93], any function computed by a *distribution* of linear threshold circuits with n^ε gates has communication complexity at most $O(n^\varepsilon \log n)$. Thus, our $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits can be simulated by communication protocols with such complexity. However, the randomized (two-party) communication complexity of IP2 is $\Omega(n)$ [CG88].

An improved targeted hitting-set generator. As mentioned above, the proofs of our results leverage the recent new approaches to derandomization. On the way to proving Theorems 1.3 and 1.4, we also make a significant contribution to the technical machinery underlying these new approaches, and this contribution is of independent interest. Specifically, a main technical ingredient in our results is a "scaled-down" version of the targeted PRG of [CT21], as follows:

Theorem 1.5 (informal; see Theorem 5.1)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}^{m(n)}$ be computable by a family of highly uniform $\mathcal{TC}^0$ circuits of size T, let $\gamma \in (0,1)$, and let $M \le T^{\Omega(\gamma)}$. Then, there exist $d' \in \mathbb{N}$ and deterministic algorithms $H_f^{\texttt{CT-TC}^0}$ and $R_f^{\texttt{CT-TC}^0}$ that for every $z \in \{0,1\}^n$ satisfy:*

1. **Generator:** *$H_f^{\texttt{CT-TC}^0}(z)$ runs in time $\mathrm{poly}(T)$ and prints a set of M-bit strings.*
2. **Reconstruction:** *$R_f^{\texttt{CT-TC}^0}(1^n)$ prints a sampler for a distribution $\mathcal{R}_f$ over $\mathcal{TC}^0_{d'} \circ \mathsf{SUM}[T^\gamma]$ oracle circuits, such that for any $D\colon \{0,1\}^M \to \{0,1\}$ that satisfies $\Pr_r[D(r)=1] \ge 1/M$ but D rejects all output strings of $H_f^{\texttt{CT-TC}^0}(z)$, we have*
$$\Pr_{R_f \leftarrow \mathcal{R}_f}\left[R_f^D(z) \text{ prints a } \mathcal{TC}^0_{d'} \text{ oracle circuit } E \text{ such that } \mathtt{tt}(E^D) = f(z)\right] \ge 2/3\,,$$
where $\mathtt{tt}(E^D)$ is the truth-table of E^D.

To compare, Chen and Tell [CT21] proved a version of Theorem 1.5 in which the function f is computable by logspace-uniform circuits of fixed polynomial depth, and the reconstruction procedure is computable by probabilistic logspace-uniform circuits of comparable depth. Achieving reconstruction with constant-depth threshold circuits requires significant technical work.

1.3 Generalizing previous characterizations of derandomization

The equivalences between refutation and derandomization generalize and strengthen previous characterizations of $pr\mathcal{BPP} = pr\mathcal{P}$, as well as allow to directly compare these characterizations. To state this, we will need a more refined technical version of Theorem 1.1.

A refinement of Theorem 1.1. As a first step, instead of refuting arbitrary non-uniform models, we consider Turing machines with non-uniform advice, and distinguish between the machine and the advice. That is, for every machine M, and every sufficiently large $n \in \mathbb{N}$, and every advice string $a \in \{0,1\}^n$, we give the refuter input (M, a) and ask it to print x such that $\Pr[M(a,x) = f(x)] \le 1/2$. We also consider the natural relaxation of refuters to **list-refuters**, in which the refuter is allowed to print a list $x_1, ..., x_{\mathrm{poly}(n)} \in \{0,1\}^n$, and it is only required that for some $i \in [\mathrm{poly}(n)]$ the string x_i will be a hard input for M with advice a.

The next two relaxations are somewhat less natural, but they make our results significantly more general. So far, the output of the hard function f depended only on the input x; we will also allow the function f to *depend on the advice a* (i.e., on the refuted algorithm), requiring that $\Pr[M(a,x_i) = f(a,x_i)] \le 1/2$ for some i. Lastly, we relax the conditions even further by considering what we call **compression list-refuters**, where we only require that $M(a,x_i)$ will fail to print a *small circuit* (say, of size $\sqrt{|f(a,x_i)|}$) whose truth-table is $f(a,x_i)$ (see Definition 3.4).

Our most general technical statement is analogous to Theorem 1.1 but holds even for the very relaxed notions of refuters described above. Let us state the result a bit informally here, while focusing for simplicity on the "refutation $\Rightarrow$ derandomization" direction:

Theorem 1.6 (informal, see Theorem 6.1)**.** *Let $\varepsilon > 0$ and $T(n) = \mathrm{poly}(n)$, and let f be any advice-dependent function that is computable in time T and hard for* str-$\mathcal{TISP}[T^{1+\varepsilon}, n^{\varepsilon}]$.[10] *Assume that there is a list-refuter in $\mathcal{FP}$ for f against* str-$\mathcal{TISP}[T^{1+\varepsilon}, n^{\varepsilon}]$ *algorithms that try to compress the output from length N to length $\sqrt{N}$. Then, $pr\mathcal{BPP} = pr\mathcal{P}$.*

For a detailed technical statement, which also includes the converse direction to Theorem 6.1 (i.e., it shows a full equivalence), see Corollary 6.5.

Generalizing and strengthening known results. Our results strictly improve over the known works that characterize $pr\mathcal{BPP} = pr\mathcal{P}$ in terms of uniform hardness hypotheses (see [CT21, LP22a, LP22a, Kor22a]). Roughly speaking, there are three "moving parts" in our equivalences between derandomization and refutation: the complexity of the hard function f, the weak class $\mathcal{C}$ of algorithms being refuted, and the complexity of the deterministic refuter itself. Ideally, we would like to deduce derandomization from refuters against the weakest-possible class $\mathcal{C}$, for any hard function $f \in \mathcal{FP}$, and while only requiring that the refuter runs in $\mathcal{FP}$.[11]

As we explain in Section 7, results in previous works [CT21, LP22a, LP22b, Kor22a] can all be recast in the terminology of refuters (see Table 1). From this perspective, all prior works relate derandomization to refuters *for the identity function*. That is, fixing a universal constant $c > 1$:

- Chen and Tell [CT21] showed that $pr\mathcal{BPP} = pr\mathcal{P}$ follows from refuters computable by logspace-uniform circuits of depth n^2 for Identity against the class $\mathcal{C}$ of probabilistic time-n^c algorithms that *only* depend on the input length (i.e., the weakest class in terms of input access). A conjecture implicit in [CT21] asserts that $pr\mathcal{BPP} = pr\mathcal{P}$ is equivalent to $\mathcal{FP}$-refuters for Identity against $\mathcal{C}$, without the depth restriction. (See Section 7.3.)

- Liu and Pass [LP22b] showed that $pr\mathcal{BPP} = pr\mathcal{P}$ is equivalent to $\mathcal{FP}$-refuters for Identity against communication protocols with runtime n^c and with n^{ε} bits of communication, for an arbitrarily small constant $\varepsilon > 0$. (Recall that this class is stronger than str-$\mathcal{TISP}[n^c, n^{\varepsilon}]$, because the communicating party is allowed arbitrary access to its input.) Korten's characterization [Kor22a] can be viewed in a similar light. (See Section 7.1.)

- Finally, the hardness assumption for conditional Kolmogorov complexity proved by Liu and Pass [LP22a] to be equivalent to $pr\mathcal{BPP} = pr\mathcal{P}$ can be viewed as a compression list-refuter for Identity against general probabilistic time-n^c algorithms. (See Section 7.2.)

Thus, the main improvement of our results (i.e., of Theorem 1.6 and Corollary 6.5) over prior work is in weakening the class of refuted algorithms (i.e., to str-$\mathcal{TISP}[T^{1+\varepsilon}, n^{\varepsilon}]$) and in extending the class of hard functions (i.e., from Identity to all functions computable in time T).

An open problem. A natural goal is to improve our results by further weakening the class of refuted algorithms, and further broadening the class of hard functions. What could be an ideal result to hope for in this context? We suggest the following open problem:

[10] Note: The class str-$\mathcal{TISP}[t(n), s(n)]$ here is defined not as non-uniform streaming algorithms, but as uniform streaming algorithms that receive non-uniform advice; see Sections 3.1 and 3.1.3 for an explanation of the distinction.

[11] There is good reason to only attempt to deduce derandomization from refuters for $f \in \mathcal{FP}$, rather than (say) relax the requirement to $f \in \mathcal{FBPP}$. Loosely speaking, a proof of the *conditional statement* "refutation of any $f \in \mathcal{FBPP}$ implies derandomization" would *unconditionally imply* that $pr\mathcal{BPP} = pr\mathcal{P}$; see Claim 6.6 for precise details.

Reference	Hard function f	Weak class $\mathcal{C}$	Refuter Complexity
[CT21]	Identity	obl-$\mathcal{BPTIME}[n^c]$	lu-$\mathcal{TIMEDEPTH}[\text{poly}(n), n^2]$
[LP22a]	Identity	$\mathcal{BPTIME}[n^c]$	$\mathcal{FP}$
[LP22b, Kor22a]	Identity	ow-$\mathcal{COMM}[n^c, n^\varepsilon]$	$\mathcal{FP}$
Thm 1.6, Cor 6.5	$\mathcal{DTIME}[n^{(1-\varepsilon)\cdot c}]$	str-$\mathcal{TISP}[n^c, n^\varepsilon]$	$\mathcal{FP}$
Conjecture	$\mathcal{FP}$	obl-$\mathcal{BPTIME}[n^c]$	$\mathcal{FP}$

Table 1: In the above, $c > 1$ is a sufficiently large universal constant, and $\varepsilon > 0$ is an arbitrarily small constant. We have the following relationships:

$$\text{obl-}\mathcal{BPTIME}[n^c] \subseteq \text{str-}\mathcal{TISP}[n^c, n^\varepsilon] \subseteq \text{ow-}\mathcal{COMM}[n^c, n^\varepsilon] \subseteq \mathcal{BPTIME}[n^c] ,$$

where obl-$\mathcal{BPTIME}[T]$ refers to probabilistic T-time algorithms that do not examine their input (i.e., only depend on its length); and ow-$\mathcal{COMM}[T, k]$ refers to probabilistic one-way communication protocols that run in time T and send k bits; and str$\mathcal{TISP}[T, S]$ refers to probabilistic streaming algorithms running in time T and space S. Also, the class lu-$\mathcal{TIMEDEPTH}[T, d]$ represents logspace-uniform circuits of size T and depth d.

Open Problem 1. *Prove the following statement, for some constant $c \geq 1$: If there is an $\mathcal{FP}$-refuter for some $f \in \mathcal{FP}$ against probabilistic algorithms running in time n^c that do not examine their input (i.e., only depend on its length), then $pr\mathcal{BPP} = pr\mathcal{P}$.*

The refuted class of algorithms in Open Problem 1 is the weakest possible in terms of the dependency on the input. Recall that if $pr\mathcal{BPP} = pr\mathcal{P}$, then (by Theorem 1.1) for any $f \in \mathcal{FP}$, and essentially any class $\mathcal{C}$ of RAMs such that there is a $\mathcal{BPP}$-refuter for f against $\mathcal{C}$, there is an $\mathcal{FP}$-refuter for f against $\mathcal{C}$. Open Problem 1 asks to prove a strong converse direction: even a refuter against the weakest possible class $\mathcal{C}$ (in terms of input-dependency) suffices to prove that $pr\mathcal{BPP} = pr\mathcal{P}$.[12] We note that an analogous statement for the case of functions $f \in \mathcal{P}$ with a single output bit is easy to prove (see Claim 3.17).

1.4 Refuters against deterministic algorithms and the Lossy Code problem

So far, we showed universal derandomization follows from (or is equivalent to) *deterministic* refuters for *probabilistic* algorithms. We show that derandomization consequences follow even from a refutation task that is potentially easier: deterministic refuters for *deterministic* algorithms.

To see this, let us recall Korten's perspective on derandomization [Kor22b], which centers around a problem called `LossyCode`. The problem is defined as follows:

Definition 1.7 (`LossyCode` [Kor22b])**.** *In* `LossyCode`*, given a pair of circuits $C\colon \{0,1\}^n \to \{0,1\}^{n-1}$ and $D\colon \{0,1\}^{n-1} \to \{0,1\}^n$ as input, the goal is to output an $x \in \{0,1\}^n$ such that $D(C(x)) \neq x$.*

[12] Indeed, Open Problem 1 asks to prove its conclusion when $f \in \mathcal{FP}$ can be arbitrary, rather than only a function that has a $\mathcal{BPP}$-refuter. However, we stated the problem in this manner only for simplicity: proving the statement in Open Problem 1 only for functions in $f \in \mathcal{FP}$ that have a $\mathcal{BPP}$-refuter would be just as interesting.

Note that `LossyCode` can be solved easily using randomness, since half of the inputs $x \in \{0,1\}^n$ satisfy the required property (and given x, it is easy to check if $D(C(x)) \neq x$). However, it seems challenging to solve the problem deterministically. In contrast to CAPP, we do not know if `LossyCode` is complete for $pr\mathcal{BPP}$, and in fact proving so would imply that $\mathcal{BPP} \subseteq \mathcal{NP}$ (see [Kor22b] for an explanation). This implies that there might be more hope for progress on deterministic poly-time algorithms for `LossyCode`, compared to CAPP.

First, we show that solving `LossyCode` reduces to (deterministically) refuting *deterministic streaming algorithms*, for any hard function in $\mathcal{FP}$. Leveraging the ideas of [Kor22b], we prove:

Theorem 1.8. *For any function $f \in \mathcal{FP}$ and $\varepsilon \in (0,1)$, if there is an $\mathcal{FP}$-refuter for f against n^{ε}-space polynomial-time deterministic streaming algorithms, then* `LossyCode` $\in \mathcal{FP}$.

To obtain a full equivalence between efficient refutation and solving `LossyCode`, we consider refuters for specific, well-studied functions. In particular, we show that solving `LossyCode` is equivalent to providing efficient refuters for `Set-Disjointness` (DISJ) or for `Inner Product` (IP) against low-space streaming algorithms, where space is measured in the number of stored bits.[13]

Theorem 1.9. *For a function $f \in \{\mathsf{DISJ}, \mathsf{IP}\}$ and all $\varepsilon \in (0,1)$, the following are equivalent:*

1. *There is a refuter in $\mathcal{FP}$ for f against n^{ε}-space poly-time deterministic streaming algorithms.*
2. *There is a refuter in $\mathcal{FP}$ for f against $(n-1)$-space poly-time deterministic streaming algorithms.*
3. `LossyCode` $\in \mathcal{FP}$.

2 Technical overview

The algorithmic framework for derandomization in this work uses targeted pseudorandom generators (tarPRGs), as defined by Goldreich [Gol11]. As in recent works [CT21, LP22a, LP22b, vMS23], we will use reconstructive tarPRGs. To describe this object, consider derandomizing the probabilistic machine $M = M^{\mathsf{CAPP}}$ that solves the $pr\mathcal{BPP}$-complete problem CAPP. At a high level,

1. Given input $x \in \{0,1\}^n$, the reconstructive tarPRG computes a string $f(x)$, and then maps $f(x)$ to a set $S_{x,f(x)}$ of n-bit strings $s_1, ..., s_{\bar{n}}$, for $\bar{n} = \mathrm{poly}(n)$. We output $\mathsf{MAJ}\,\{M(x, s_i)\}_{i \in [\bar{n}]}$.
2. The pseudorandomness of $S_{x,f(x)}$ for $M(x, \cdot)$ follows by designing an efficient reconstruction algorithm R: Assuming that $\Pr_{r \in \{0,1\}^n}[M(x,r) = 1] \notin \Pr_{i \in [\bar{n}]}[M(x, s_i) = 1] \pm 1/10$, the algorithm $R^{M(x,\cdot)}$ computes $x \mapsto f(x)$ "too efficiently". Since our hypothesis will be that f is hard to compute very efficiently on x, we reach a contradiction.

In recent works, the mapping of $f(x)$ to the set $S_{x,f(x)}$ generally used known technical tools: for example, we may think of $f(x)$ as the truth-table of a function $\{0,1\}^{\log(|f(x)|)} \to \{0,1\}$, and apply the Nisan-Wigderson construction [NW94] (with the code of [STV01]) to this function. The

[13] In the DISJ_n (IP_n resp.) problem, one is given two n-bit strings $x, y \in \{0,1\}^n$ (y is given after all of x) and the goal is to determine whether their inner product $\sum_{i=1}^{n} x_i y_i$ is non-zero (odd resp.).

novelty in [LP22a, LP22b], following [CT21, Section 2.1], was in reanalyzing the known reconstruction argument of [NW94, IW98, STV01] to prove the correctness of the tarPRG, applying the same high-level template outlined above, with a suitable (new) hardness assumption.[14]

For example, if the reconstruction R requires n^{ε} queries to the truth-table $f(x)$ (as in [NW94, IW98, STV01]), then one needs to assume that the mapping $x \mapsto f(x)$ is hard to compute even if one is allowed "leakage" of n^{ε} bits from $f(x)$. Furthermore, if we want the tarPRG to succeed on all inputs, then this same type of hardness should hold for all (but at most finitely many) inputs. This is precisely how the result in [LP22b] is proved.

2.1 Our starting point: A new perspective

We suggest a new perspective on the above framework. Let us think of the problem of computing f *algorithmically*: how hard it is to compute f that will have the required properties? Another way to frame this question is to ask: *given x, how hard is it to find $f(x)$ such that $R^{M(x,\cdot)}(x)$* **fails** *to print $f(x)$, when it has some "limited access" to $f(x)$?* (The meaning of "limited access" here could be, say, n^{ε} bits of information, as in [LP22b].)

Our key observation is to think of x not as specifying an input, but rather as specifying the algorithm $R_x = R^{M(x,\cdot)}(x)$, and to think of $f(x)$ not as the *output* of R_x, but rather as a potential *input* for R_x. That is, reformulating the question above:

> *We are given a description of an algorithm R_x, and our task is to find a string y such that R_x fails to print y, even when R_x has some "limited access" to y.*

Indeed, this is precisely a refutation task for the algorithm R_x, where we are trying to find a "bad" input y demonstrating that R_x does not compute the hard function $\mathsf{Identity}(y) = y$. Moreover, recall that in previous works, the requirement was that computing the mapping $x \mapsto y$ will be hard for the reconstruction algorithm R on all but finitely many x. From the current viewpoint, this translates into requiring that the refuter will *succeed in the worst-case*, i.e., succeed in finding a hard y when given any R_x (except, perhaps, on finitely many x).

From a technical viewpoint, given the perspective above, we will improve the known results by: (1) extending the class of hard functions (i.e., allowing more hard functions than just $\mathsf{Identity}$), and (2) creating more efficient reconstruction algorithms R, such that they can work with "even less access" to y, and with more restricted computational resources.

2.2 Warm-up: The Nisan-Wigderson generator

As a warm-up, let us prove that (deterministic polynomial-time) refuting the function $\mathsf{Identity}$ against streaming algorithms with n^{ε} space (for an arbitrarily small constant $\varepsilon > 0$) implies $pr\mathcal{BPP} = pr\mathcal{P}$.

We are given x as input to the probabilistic algorithm $M = M^{\mathsf{CAPP}}$ which solves CAPP. We know in advance the reconstruction algorithm R that our proof will use (see below), and moreover there is an efficient mapping from x to $R_x = R^{M(x,\cdot)}$. Therefore we can compute the description of R_x, and feed the description to the poly-time refuter, which outputs y. Thinking

[14] Loosely speaking, the original argument of [IW98] applied only to functions that have certain structural properties (i.e., are downward self-reducible and randomly self-reducible), yet required standard hardness assumptions. In [CT21, Section 2.1] and [LP22a, LP22b] it was reanalyzed (for tarPRGs) without the assumption that the function has structural properties, but with new types of hardness assumptions.

of y as a truth-table, we use the standard construction of [NW94, IW98, STV01] to obtain a set of strings that is hopefully pseudorandom.

The main observation needed for the proof is that the reconstruction algorithm R of [NW94, IW98, STV01] can be implemented by a *streaming algorithm that passes over* y. Specifically, the combination of the local list-decoder of [STV01] and of the reconstruction of [NW94, IW98] only requires making non-adaptive linear queries to y (since the code of [STV01] is linear, and since the queries of [NW94, IW98] are non-adaptive). Indeed, a streaming algorithm can first toss random coins to choose linear queries to y, then resolve these queries in a single pass over y, and finally run the rest of the reconstruction procedure without accessing y again.

Furthermore, this streaming algorithm also uses *low space*. This essentially follows by a padding argument: given $x \in \{0,1\}^{n_0}$, we instantiate the argument above with $x' = x0^{n-n_0}$, where $n = (n_0)^{C/\varepsilon}$ for a sufficiently large constant $C > 1$. The number of coins that M needs is $|x| = n^{\varepsilon/C}$, and therefore (closely inspecting the reconstruction argument in [NW94, IW98, STV01] for this parameter setting) the number of queries to y is at most, say, $n^{\varepsilon/2}$. Thus, the streaming algorithm only needs n^{ε} space to resolve these queries during its pass on y.

2.3 A broader class of hard functions

Let us now describe the proof of Theorem 1.1. The main part of the proof is to deduce derandomization from the existence of a refuter for any function f computable in time $T(n) = \mathrm{poly}(n)$ against streaming algorithms running in time $T^{1+\varepsilon}$ and space n^{ε}.

Starting with the argument above, instead of applying the PRG construction of [NW94, IW98, STV01] to y, we will apply a targeted hitting-set generator (tarHSG) H^{CT} from [CT21] to y, where H^{CT} is instantiated with the hard function f. That is, given x, we first compute a description of $R_x = R^{M(x,\cdot)}$ for a predetermined reconstruction algorithm R that will be presented below, run the refuter on R_x to obtain a bad input y, and finally run H^{CT}, instantiated with the hard function f, on input y, to obtain a pseudorandom set.

We argue that this construction is a tarHSG,[15] which implies that $pr\mathcal{RP} = pr\mathcal{P}$ and hence (by [Sip83, Lau83, BF99, GZ11]) $pr\mathcal{BPP} = pr\mathcal{P}$. To do so, we analyze H^{CT} in a different way than in [CT21]. Recall that for any f computable in deterministic time T and input y for f, the generator H^{CT} produces $t \approx T$ sets $S^{(1)}_{f,y}, ..., S^{(t)}_{f,y}$.[16] We argue that the following holds: If $M(x,\cdot)$ distinguishes every set $S^{(i)}_{f,y}$ from random, then we can compute $y \mapsto f(y)$ by a one-pass streaming algorithm R_x using time $T^{1+\varepsilon}$ and space n^{ε}. Since this contradicts the properties of the refuter (i.e., the refuter finds y that fails R_x), we conclude that our construction is indeed a tarHSG.

To prove this we need to give a reconstruction algorithm R_x with such properties. We recall the following facts about H^{CT} and about its known reconstruction algorithm:

1. The generator H^{CT} simulates the uniform circuit computing $f(y)$, and transforms the matrix $\mathcal{G}^{(y,f)}$ representing the gate-values in this circuit into an "encoded" matrix $\mathcal{B}^{(y,f)}$ that we call a bootstrapping system, which has useful properties (the transformation uses the ideas

[15]That is, if $\Pr_x[M(x,r) = 1] \geq 1/2$ then there exists a string s in the pseudorandom set such that $M(x,s) = 1$.

[16]The original work [CT21] required that f will be compuable by logspace-uniform circuits of size T and depth d, and the number of pseudorandom sets was $t \approx d$. In this work we use any function computable in time T, and instantiate the original construction with $d \approx T$ (as any function computable in time T is computable by logspace-uniform circuits of size $\tilde{O}(T)$ and depth $\tilde{O}(T)$).

of Goldwasser, Kalai, and Rothblum [GKR15]). For simplicity, assume that the dimensions of $\mathcal{B}^{(y,f)}$ are identical to those of $\mathcal{G}^{(y,f)}$. Then, H^{CT} applies the generator of [NW94] to each of the $t \approx T$ rows of $\mathcal{B}^{(y,f)}$, to obtain pseudorandom sets $S^{(1)}_{f,y}, ..., S^{(t)}_{f,y}$. The output is $\cup_i S^{(i)}_{f,y}$.

2. The reconstruction argument works in a layer-by-layer fashion: it starts from the bottom layer, which has an encoding of y, and in the end reaches the top layer, which has $f(y)$. For each layer $i = 1, ..., t$ sequentially, we run the reconstruction argument of [NW94, IW98] R^{NW} to obtain a small circuit C_i whose truth-table is the i^{th} layer. The algorithm R^{NW} needs to make queries to the $(i-1)^{th}$ layer,[17] and since we already have a circuit C_{i-1} whose truth-table is the $(i-1)^{th}$ layer, we can simulate C_{i-1} to answer the queries of R^{NW}.

As in Section 2.2, since the number of random coins that we need is, say, $n^{\varepsilon/2}$, each step of the reconstruction can be executed in time n^{ε} (and in particular, each step makes at most n^{ε} queries and prints a circuit of size at most n^{ε}). This yields an algorithm R_x that computes $y \mapsto f(y)$ in time $T^{1+\varepsilon}$, but we still have not explained why R_x is a low-space one-pass streaming algorithm.

The key observation is that we can implement R_x with limited access to y. Specifically, we start the reconstruction from the *second layer* of the circuit for $f(y)$. The only time we need access to the first layer, which encodes y, is when answering the queries of R^{NW} to the first layer (i.e., when we run R^{NW} to get a circuit C_2 for the second layer). Moreover, since the first layer is a *linear* encoding of y, to answer these queries we only need to compute linear functions of y. Since there are at most n^{ε} queries in each step, we can compute these n^{ε} linear functions of y by an $\approx n^{\varepsilon}$-space one-pass streaming algorithm. For precise details, see Theorem 3.15 and Section 6.1.

The converse direction: Obtaining an equivalence. To prove Theorem 1.1 we also need to show the converse direction, i.e., that derandomization implies refutation. Observe that the first direction (described above) holds for any f in time T; to get an equivalence, we now restrict our attention to f's that have a $\mathcal{BPP}$-refuter, denoted Ref_f.

Then, proving the converse direction is simple. Note that we can *test* whether a given string y is actually a bad string for R_x (i.e., by computing $f(y)$, simulating $R_x(y)$, and comparing the outcomes). Thus, to find y that will be bad for R_x, we run a search-to-decision reduction as in [Gol11]: we construct random coins for Ref_f bit-by-bit, and in each step we verify that the probability that Ref_f outputs a string that is bad for R_x (conditioned on the current prefix of coins) is approximately maintained. Each step requires solving a decision problem in $pr\mathcal{BPP}$, and thus (by our assumption) this problem can be solved in $pr\mathcal{P}$. For details see Theorem 6.4.

2.4 Extending the connection down to $\mathcal{TC}^0$, and an improved Chen-Tell generator

Next, we prove that the equivalence between refutation and derandomization is more general, and in fact scales all the way down to $\mathcal{TC}^0$ circuits. The equivalence stated in Theorem 1.2, which refers to the specific hard function $\mathsf{Identity}$, follows from ideas similar to the ones in Section 2.2, only with a more careful analysis of the known algorithms of [NW94, IW98, STV01] (for details see Theorem 3.14, Appendix B, and Theorem 6.7).

[17] This description abstracts away many technical details. For example, the algorithm R^{NW} actually needs to make queries to the i^{th} layer to construct C_i. We require $\mathcal{B}^{(f,y)}$ to be downward self-reducible, and thus these queries can be answered by a small number of queries to the $(i-1)^{th}$ layer. (The other property that we require from $\mathcal{B}^{(f,y)}$ is that each layer will be a codeword in a sufficiently good error-correcting code; see Section 5 for details.)

We therefore focus on the connection in Theorem 1.4, whose proof is the most technically involved part of this work. Let us first sketch the proof of the special case stated in Theorem 1.3: if there is a refuter for any function in highly uniform $\mathcal{TC}^0$ against distributions over $\mathcal{TC}^0 \circ \mathsf{SUM}$ of size n^ε, then $\mathsf{CAPP}_{0,1/2}$ of $\mathcal{TC}^0$ circuits can be solved in deterministic polynomial time.

The $\mathsf{CAPP}_{0,1/2}$ algorithm is similar to the one in Section 2.3: it receives an input $x \in \{0,1\}^n$ (which represents a $\mathcal{TC}^0$ circuit of size n^ε), computes the description of a sampler $R_x = S^x$ for a distribution over $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits (where S is a predetermined uniform algorithm that we describe below), feeds R_x into the refuter to obtain y, and runs a tarHSG $H^{\texttt{CT-TC}^0}$ that we will construct (instantiated with the function f) on input y to obtain pseudorandom strings.

Our goal is to construct $H^{\texttt{CT-TC}^0}$ that is instantiated with a function f computable by highly uniform $\mathcal{TC}^0$ circuits of size $T(n) = \mathrm{poly}(n)$, such that $H^{\texttt{CT-TC}^0}$ has a reconstruction algorithm Rec that is a distribution over $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits with n^ε gates. To do so, consider the matrix $\mathcal{G}^{(f,y)}$ of gate-values for $f(y)$, which has $d = O(1)$ rows and T columns. We want to encode $\mathcal{G}^{(f,y)}$ into a bootstrapping system $\mathcal{B}^{(f,y)}$ that has a $\mathcal{TC}^0 \circ \mathsf{SUM}$ reconstruction Rec, as follows:

1. For $d' = O(d)$, every circuit in the support of R_x will consist of a sequence of $d' - 1$ $\mathcal{TC}^0$ circuits $\mathsf{Rec}^{(2)}, ..., \mathsf{Rec}^{(d)}$ of size n^ε, where $\mathsf{Rec}^{(i)}$ corresponds to the i^{th} row of $\mathcal{B}^{(f,y)}$.

2. For $i = 2, ..., d'$, the circuit $\mathsf{Rec}^{(i)}$ gets access to a distinguisher D for the tarHSG (we think of D as the $\mathcal{TC}^0$ circuit x), and prints a circuit C_i whose truth-table is the i^{th} layer in $\mathcal{B}^{(f,y)}$; to do so, $\mathsf{Rec}^{(i)}$ makes non-adaptive queries to C_{i-1} (i.e., to the circuit that $\mathsf{Rec}^{(i-1)}$ printed).

3. The circuit C_1 (that $\mathsf{Rec}^{(2)}$ queries) consists of a layer of n^ε "SUM gates" such that each "gate" computes a weighted sum (over the integers) of the bits of y.[18]

The technical challenges, and our high-level approach. The reconstruction algorithm for each row in [CT21] is an $\mathcal{NC}$ circuit. We do not know how to design a more efficient reconstruction algorithm (in particular, in $\mathcal{TC}^0$) for each row when using their $\mathcal{B}^{(f,y)}$: the reason is that the reconstruction algorithm implements the list-decoder for the Reed-Muller code [STV01], which in turn uses the list-decoder for the Reed-Solomon code [Sud97]; there is currently no known list-decoder for Reed-Solomon that works in constant depth.

To support $\mathcal{TC}^0$ reconstruction of each row, we will construct a new bootstrapping system $\mathcal{B}^{(f,y)}$. This bootstrapping system can be viewed as a new and more efficient version of the [GKR15] encoding of uniform circuits. For an arbitrarily small constant $\delta > 0$, the bootstrapping system $\mathcal{B}^{(f,y)}$ has $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuits of size T^δ that can list-decode each row from distance $1/2 + T^{-\Omega(\delta)}$, and that reduce the computation of an entry in a row i to the computation of T^δ entries in row $i-1$. (See Proposition 5.5 for a precise statement.)

The main technical ingredient in the construction of $\mathcal{B}^{(f,y)}$ is an error-correcting code that is *locally encodable* and *approximately locally decodable* by uniform $\mathcal{TC}^0$ circuits; that is:

Proposition 2.1 (informal, see Proposition 4.1)**.** *For every $\gamma, \nu > 0$ and finite field $\mathbb{F}$ of size $|\mathbb{F}| \leq \mathrm{poly}(N)$ there exists a mapping $\mathsf{Enc}\colon \mathbb{F}^N \to \{0,1\}^{\bar{N}}$, where $\bar{N} = N^{c_{\gamma,\nu}}$, such that the following holds:*

1. **(Locally encodable.)** *There is a $\mathcal{P}$-uniform family of $\mathcal{TC}^0$ circuits of size $N^{O(\gamma+\nu)}$ that gets input $i \in [\bar{N}]$, queries $z \in \mathbb{F}^N$ at N^γ locations, and outputs $\mathsf{Enc}(z)_i$.*

[18] We write "gate" because this functionality is implemented in binary, and therefore each "SUM gate" actually consists of several gates, which represent the outcome of the weighted sum in binary.

2. **(Locally approximately decodable.)** *There is a $\mathcal{P}$-uniform family $\{D_N\}_{N\in\mathbb{N}}$ of probabilistic oracle $\mathcal{TC}^0$ circuits of size $N^{O(\gamma+\nu)}$ such that for every $z \in \mathbb{F}^N$ and any $O \in \{0,1\}^{\bar{N}}$ satisfying $\Pr_{j\in[\bar{N}]}\left[\mathsf{Enc}(z)_j = O(j)\right] > 1/2 + N^{-\nu}$, the following holds. The circuit D_N first has a probabilistic preprocessing step, in which it non-adaptively queries z, and with probability $1-o(1)$ satisfies the following. There is $S \subseteq [N]$ of density $|S|/N \geq 1 - N^{-\gamma}$ such that for every $i \in S$,*

$$\Pr\left[(D_N)^O(i) = z_i\right] > 2/3\,,$$

where the probability is over the random coins of D_N after the preprocessing step.

We believe that the improved bootstrapping system and the code in Proposition 2.1 are of independent interest, and may find further applications. As one example, they allow us to scale down the results in [CT21] to hold for $\mathcal{TC}^0$ circuits, rather than only for $\mathcal{NC}$ circuits; this is essentially the content of Theorem 1.5 (see Theorem 5.1 for the detailed statement).

2.4.1 The new bootstrapping system: an improved version of the GKR encoding

The idea for constructing $\mathcal{B}^{(y,f)}$ in [GKR15, CT21] is to think of each row $i \in [d]$ in $\mathcal{G}^{(f,y)}$ as a function $\alpha_i\colon \{0,1\}^{\log(T)} \to \{0,1\}$, arithmetize the row as a polynomial $\hat{\alpha}_i\colon \mathbb{F}^m \to \mathbb{F}$, and insert additional polynomials between each pair of rows that implement a sumcheck-like functionality. This yields a matrix (with entries in $\mathbb{F}$) such that each row is a codeword in a locally list-decodable code, and computing any entry in row i efficiently reduces to computing a few entries in row $i-1$ (the reader is referred to [CT21] for a detailed explanation).

Our goal is to construct $\mathcal{B}^{(y,f)}$ when f is a highly uniform $\mathcal{TC}^0$ circuit, such that the local list-decoder for each row is a $\mathcal{TC}^0$ circuit, and the downward reduction from row i to row $i-1$ is computable in $\mathcal{TC}^0$.

Arithmetization and sumcheck polynomials. We first define α_i differently than in [GKR15, CT21]: for every threshold gate $g(y) = \mathbf{1}[\sum_h w_h \cdot h(y) > \theta_g]$ (where the h's are the gates feeding into g, and the $w_{g,h}$'s and θ_g are real numbers), we define $\alpha_i(g) = \sum_h w_{g,h} \cdot h(y)$. The arithmetization of α_i is now straightforward, i.e. $\hat{\alpha}_i(g) = \sum_h \hat{\Phi}(g,h) \cdot h(y)$ where $\hat{\Phi}$ is an appropriate arithmetization of the function $\Phi(g,h) = w_{g,h}$ (see below). Whenever our algorithms (e.g., for downward self-reducibility) will need to obtain a value $g(y)$ in the i^{th} layer given access to $\hat{\alpha}_i$, they will compute the function $\mathbf{1}[\hat{\alpha}_i(g) > \theta_g]$, which can be done in $\mathcal{TC}^0$.

Relying on the fact that the size-T circuit for f is highly uniform (which means that Φ is computable by a uniform $\mathcal{TC}^0$ circuit of size $T^{o(1)}$; see Definition 3.6), we arithmetize the Φ's by polynomials of degree T^δ over a field of size $p = \Theta(T^2)$, where $\delta > 0$ is a sufficiently small constant. This allows us to insert only constantly many sumcheck-like polynomials between each pair of rows, and hence $\mathcal{B}^{(f,y)}$ is of constant depth $d' = O(d)$. (See Proposition 5.3 for details.)

Now we have a sequence of d' rows such that each row is a codeword in the Reed-Muller code, and the sequence is downward self-reducible by uniform $\mathcal{TC}^0$ circuits (again, details appear in Proposition 5.3). The main trouble is that the local list-decoder for each row, i.e. the local list-decoder for the Reed-Muller code, is not known to be in $\mathcal{TC}^0$.

Local encodability and approximate local decodability for reconstruction. In [CT21], each row was further encoded by the Hadamard code to yield a binary matrix $\mathcal{B}^{(f,y)}$ (whose rows were used as truth-tables for the generator of [NW94]). To resolve the problem above, instead of the Hadamard code, we encode each row by the code from Proposition 2.1.

To see why this is helpful, think of each $\hat{\alpha}_i$ as already encoded in a code that is *uniquely decodable in* $\mathcal{TC}^0$ from distance $1 - T^{-\Omega(1)}$: the $\mathcal{TC}^0$ decoder implements the standard unique decoding for the Reed-Muller code. Combined with the $\mathcal{TC}^0$ local approximate decoder of the code from Proposition 2.1, each row is now locally decodable from agreement $1/2 + T^{-\Omega(1)}$ by $\mathcal{TC}^0$ circuits, as we wanted.

To prove that $\mathcal{B}^{(f,y)}$ is still downward self-reducible, we will rely on the $\mathcal{TC}^0$-local-encoding property of the code. Specifically, since each entry j in row i is a local encoding of $\hat{\alpha}_i$, computing the j^{th} entry reduces to computing "a few" values of $\hat{\alpha}_i$; and computing each value of $\hat{\alpha}_i$ reduces to computing "a few" values of $\hat{\alpha}_{i-1}$, which in turn appear as entries in the encoding of $\hat{\alpha}_{i-1}$.[19] And since the local encoding of the code is computable in $\mathcal{P}$-uniform $\mathcal{TC}^0$ of size T^{δ}, this sequence of reductions can be computed in $\mathcal{P}$-uniform $\mathcal{TC}^0$ of such size.

The last part is implementing the base case, i.e. the bottom row of $\mathcal{B}^{(f,y)}$. This bottom row needs to compute values of the low-degree extension of y (since these are the queries made by the downward self-reducibility algorithm, when running the reconstruction for the second row). Indeed, these values can be computed using SUM gates (see Proposition 5.3 for details).

2.4.2 A $\mathcal{TC}^0$-locally encodable and $\mathcal{TC}^0$-locally approximately-decodable efficient code

The proof of Proposition 2.1 follows a recent construction of a code by Doron and Tell [DT23].[20] The code is actually a combination of two codes: the first code increases the distance from $N^{-\Omega(1)}$ to a tiny constant $\delta > 0$, using a refinement of a construction by Goldwasser *et al.* [GGH+07]; and the second code increases the distance from δ to $1/2 - N^{-\Omega(1)}$, using the derandomized direct product of Impagliazzo and Wigderson [IW97].

The first code. We use the classical expander-based distance-amplification of Alon *et al.* [ABN+92], to increase the distance from $N^{-\Omega(1)}$ to (say) 0.4. This code has a constant-depth decoder, and as proved by Gutfreund and Viola [GV04] (see [GGH+07]), using the Gabber-Galil [GG79] expander, encoding can be done by constant-depth circuits (see Lemma 4.4).

The problem is that now the alphabet is large, and we want to decrease it to binary. Moreover, we want to do so while maintaining a non-adaptive constant-depth decoder, since non-adaptivity is important for the construction of $\mathcal{B}^{(f,y)}$. An idea from [DT23] is to use a sequence of concatenation steps with different codes to gradually decrease the alphabet, while approximately maintaining the distance and preserving the complexity of the decoder at each step. We follow the same approach, while ensuring local-encodability in $\mathcal{TC}^0$ (see Sections 4.1.2 and 4.1.3).

The second code. We use the derandomized direct-product code of [IW97], concatenated with the Hadamard code, to increase the distance from δ to $1/2 - N^{-\Omega(1)}$. Indeed, this code is locally encodable by $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuits; to see this, let us focus on local encodability of the code of [IW97]. Given an output index i, we can compute the locations in the input that appear in the

[19] Indeed, while we did not state this in Proposition 2.1, the code is systematic; see Proposition 4.1.

[20] We use the same ideas as in [DT23], but cherry-pick parts of the construction, and argue different properties.

i^{th} output location, by XORing: (1) the output of the expander-random-walk sampler (we again use the Gabber-Galil expander, which is computable in constant depth), and (2) the output of a combinatorial design function (where the combinatorial design is hard-wired into the circuit by the $\mathcal{P}$-uniform algorithm constructing the circuit). See Proposition 4.6 and Claim 4.6.1 for details.

The local decodability of this code by $\mathcal{TC}^0$ circuits is presented in a non-standard way in Proposition 2.1, but it (essentially) already follows from a close examination of the decoding algorithms from [GL89, IW97]. See the proof of Proposition 4.6 for details.

2.4.3 Getting a near-equivalence: a $\mathcal{TC}^0$-samplable reconstruction

So far, we described the proof of Theorem 1.3, which asserts that efficient refutation of distributions over small $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits implies derandomization of $\mathcal{TC}^0$ (with one-sided error). To prove the two-way connection stated in Theorem 1.4, we need an additional observation.

Recall that in the argument above, we denoted by Rec the distribution over small $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits, and we also mentioned that Rec has a uniform sampler S. However, the argument already supports a stronger statement: going through our proofs, we can *implement S as a $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit* (of fixed polynomial size, say n^2). It follows that $R_x = S^x$ is a $\mathcal{TC}^0$ circuit that samples a distribution over $\mathcal{C}$, where $\mathcal{C}$ is the class of small $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits.

This observation paves the way towards proving a converse direction, i.e., showing that derandomization of $\mathcal{TC}^0$-samplable distributions over $\mathcal{C}$ implies refutation of such distributions. To see this, assume that we have a deterministic polynomial-time CAPP algorithm for $\mathcal{TC}^0$, and let f be a function with a $\mathcal{TC}^0$-refuter (as detailed in the hypothesis of Theorem 1.4). Given a $\mathcal{TC}^0$-sampler for a distribution over $\mathcal{C}$, we use the same search-to-decision reduction as in Section 2.3: we construct random coins for the refuter bit-by-bit, where the decision at each step reduces to solving CAPP for $\mathcal{TC}^0$. For the full details, see Theorem 6.13.

3 Preliminaries

For a positive integer k, we use $[k]$ to denote the set $\{1, 2, \ldots, k\}$. We use $\mathbb{N}$ to denote all non-negative integers and $\mathbb{N}_{\geq 1}$ to denote all positive integers.

As mentioned in Section 1, in this paper we consider refuters for non-uniform models of computation. We will have two formalizations of non-uniform models: the first refers to RAMs that take advice, and is presented in Section 3.1; and the second refers to non-uniform circuits, and is presented in Section 3.2.

3.1 Classes of RAMs, and refuters for machines with advice

The machine model in this paper is the RAM model, and in particular we consider classes of RAMs that take advice. More formally, these will be RAMs that take two inputs (a, x), and we think of a as non-uniform advice and of x as the actual input, and analyze the machine accordingly (see Section 3.1.3). Throughout the paper, when referring to such machines, we will usually omit the suffix "that takes advice", but this is always implicitly assumed.

3.1.1 Streaming algorithms

One class of RAMs that we will repeatedly refer to in the paper is streaming algorithms (that take advice), defined as follows:

Definition 3.1 (streaming algorithms)**.** *A* one-pass streaming algorithm running in time T and in space s *is a RAM that takes as input* (a, x)*, runs in time* $T(|a| + |x|)$ *and in space* $s(|a| + |x|)$*, and accesses* x *in a bit-by-bit fashion, reading each bit of* x *once and in-order. (There is no limitation as to how the machine accesses* a*.) We denote the class of such algorithms by* str-$\mathcal{TISP}[T, s]$.

Recall that in the beginning of Section 1 we referred to str-$\mathcal{TISP}$ as the class of *non-uniform* streaming algorithms, rather than as the class of uniform streaming algorithms that take advice. We explain this difference in Section 3.1.3.

3.1.2 Refuters for classes of RAMs

To define refuters for classes of RAMs, we consider a generalized notion of a hard function, in which the function may also depend on the advice. More formally:

Definition 3.2 (algorithm-dependent hard function)**.** *Let* $f: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *and let* $\mathcal{C}$ *be a class of probabilistic RAM machines, and let* $p: \mathbb{N} \to \mathbb{N}$*. We say that* f *is a* p-bounded algorithm-dependent hard function against $\mathcal{C}$ *if for every* $M \in \mathcal{C}$ *and sufficiently large* $n \in \mathbb{N}$ *and string* $a \in \{0,1\}^n$ *there exists* $x \in \{0,1\}^{p(n)}$ *such that* $\Pr[M(a, x) = f(a, x)] < 2/3$.

A refuter for a class $\mathcal{C}$ gets as input a description of $M \in \mathcal{C}$ and also an arbitrary advice a, and outputs x such that $M(a, x)$ fails to compute $f(a, x)$. The first type of refuter that we define is a list-refuter, which outputs a set $x_1, ..., x_t$ such that for some $i \in [t]$ it holds that $M(a, x_i)$ fails to compute $f(a, x_i)$.

Definition 3.3 (list-refuter)**.** *Let* $\mathcal{C}$ *be a class of probabilistic RAM machines, and let* f *be a* p*-bounded algorithm-dependent hard function against* $\mathcal{C}$ *for some* p*. An algorithm* A *is a* $\mathcal{P}$-computable list-refuter for $\mathcal{C}$ against f *if for every* $M \in \mathcal{C}$ *and sufficiently large* $n \in \mathbb{N}$*, when given as input the description of* M *and a string* $a \in \{0,1\}^n$*, the algorithm* A *runs in deterministic time* $\text{poly}(n)$ *and prints a length-*t *list* $x_1, \ldots, x_t \in \{0,1\}^{p(n)}$ *such that for some* $i \in [t]$ *it holds that*

$$\Pr\left[M(a, x_i) \textit{ prints } f(a, x_i)\right] < 2/3 \,.$$

We say A *is a* refuter *if the length of all output lists is always* 1.

The next notion of refuter is more relaxed: we ask the refuter again to output $x_1, ..., x_t$, but this time we only require that for some $i \in [t]$ it holds that $M(a, x_i)$ fails to compute a *compressed version* of $f(a, x_i)$, in the form of a small circuit whose truth-table is $f(a, x_i)$.

Definition 3.4 (compression list-refuter)**.** *Let* $\mathcal{C}$ *be a class of probabilistic RAM machines, and let* f *be a* p*-bounded algorithm-dependent hard function against* $\mathcal{C}$ *for some* p*. An algorithm* A *is a* $\mathcal{P}$-computable s-compression list-refuter for $\mathcal{C}$ against f *if for every* $M \in \mathcal{C}$ *and sufficiently large* $n \in \mathbb{N}$*, when given as input the description of* M *and a string* $a \in \{0,1\}^n$*, the algorithm* A *runs in deterministic time* $\text{poly}(n)$ *and prints a length-*t *list* $x_1, \ldots, x_t \in \{0,1\}^{p(n)}$ *such that for some* $i \in [t]$ *it holds that*

$$\Pr\left[M(a, x_i) \textit{ prints a circuit of size } s(|a| + |x_i|) \textit{ whose truth-table is } f(a, x_i)\right] < 2/3 \,.$$

We say A *is an* s*-compression refuter if the length of all output lists is always* 1.

Note that the circuit size s in Definition 3.4 is a function of the input length to f (i.e., of $|a| + |x|$), rather than a function of the length of the truth-table $|f(a,x)|$. One may think of this as compressing the input (a,x) such that the compressed version still contains enough information to efficiently produce the output $f(a,x)$.

The next notion of refuters is *randomized refuters*, which tosses random coins, and with noticeable probability prints a string x such that $M(a,x)$ fails to compute $f(a,x)$. (In this definition we will not use the relaxations of list-refuters and of compression refuters.)

Definition 3.5 (randomized refuters). *Let $p\colon \mathbb{N} \to \mathbb{N}$, let $\mathcal{C}$ be a class of probabilistic RAM machines, and let $f\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a p-bounded algorithm-dependent hard function against $\mathcal{C}$. We say that f admits a* polynomial-time randomized refuter against $\mathcal{C}$*, if there exists a randomized algorithm B and a polynomial q such that for every $M \in \mathcal{C}$ and sufficiently large $n \in \mathbb{N}$ and string $a \in \{0,1\}^n$, with probability at least $1/q(n)$, $B(M,a)$ outputs a length-$p(n)$ string x satisfying $\Pr[M(a,x) = f(a,x)] < 2/3$.*

3.1.3 Non-uniform classes of RAMs

In Theorem 1.1, we considered what we referred to there as non-uniform classes of algorithms, where for every input length n, the class contains a set $\mathcal{C}_n$ of probabilistic algorithms whose description is of length n and that are executed on inputs of length n.

This presentation in Theorem 1.1 was done merely for simplicity. The formalization of non-uniform classes of RAMs does not explicitly appear in our technical results, since our technical results use the more refined notion presented in this section, which separates a machine $M \in \mathcal{C}$ from the advice $a \in \{0,1\}^*$ that it gets.[21] However, the refined formalization does capture the notion of non-uniform algorithms. For example, to capture any streaming algorithm C of description length n, we can fix $\mathcal{C}$ to contain a universal machine U that intreprets its input as a description of a streaming algorithm, and let $a \in \{0,1\}^n$ be a description of C.

3.2 Classes of circuits, and refuters for circuits

For convenience, we consider circuit families with many input parameters. Specifically, a circuit family with k input parameters $\vec{\ell} = (\ell_1, \ell_2, \ldots, \ell_k) \in \mathbb{N}^k$ is defined as $\{C_{\vec{\ell}}\}_{\vec{\ell} \in \mathbb{N}^k}$. We say that a circuit family $\{C_{\vec{\ell}}\}_{\vec{\ell} \in \mathbb{N}^k}$ is $\mathcal{P}$-uniform if there is an algorithm A_C that, given input parameters $\vec{\ell} \in \mathbb{N}^k$, outputs the description of $C_{\vec{\ell}}$ in $|C_{\vec{\ell}}|$ time.

3.2.1 Threshold Circuits

Notation. Consider a family of threshold circuits of depth $d = d(n)$ and with $T = (n)$ gates. For any $n \in \mathbb{N}$ and $i \in [d]$ and $j \in [T]$, denote by $g_{i,j}$ the j^{th} gate in the i^{th} layer, and denote the function that $g_{i,j}$ computes by

$$g_{i,j}(x) = \mathbf{1}\left[\sum_{k \in [T]} w_{i,j,k} \cdot g_{i-1,k}(x) > \theta_{i,j}\right] ,$$

[21] Implicitly, the machine's description is of constant size, since in our formalization we first fix the machine and then consider an advice a that is arbirarily long.

where $\theta_{i,j} \in \mathbb{Z}$ and $w_{i,j,k} \in \mathbb{Z}$ for all $k \in [T]$. Denoting $W = \max_{i,j,k} \{w_{i,j,k}\}$, we assume throughout the paper that $W \le T$. We also assume, without loss of generality, that $|\theta_{i,j}| \le T^2$.

We denote by $\mathcal{TC}^0 \circ \mathsf{SUM}$ the class of families of constant-depth circuits with threshold gates such that for every family there exists a constant $C > 1$ for which the following holds. Each circuit in the family has a layer of gates at the bottom, where the gates in the layer are partitioned into blocks of size $(C+1) \cdot \log(n)$, and each block computes a weighted sum of the inputs (represented in binary) over the integers, with weights bounded by n^C.

For $S\colon \mathbb{N} \to \mathbb{N}$, we use $\mathcal{TC}^0_d\text{-}\mathsf{WIRES}[S]$ to denote the class of depth-d threshold circuits with at most S *wires* (instead of gates). We also use $\mathcal{TC}^0_d\text{-}\mathsf{WIRES}[S] \circ \ell\text{-}\mathsf{XOR}$ to denote a circuit consists with a top $\mathcal{TC}^0_d$ circuit of S total wires and a bottom layer of ℓ parity gates. Similarly for $\mathcal{TC}^0_d\text{-}\mathsf{WIRES}[S] \circ \ell\text{-}\mathsf{SUM}$.

Highly uniform circuits. The following definition of highly uniform threshold circuits is a more precise and fine-grained version of the definition that appeared in Section 1.2.

Definition 3.6 (highly uniform threshold circuits). *Let $T, d\colon \mathbb{N} \to \mathbb{N}$, and let $\delta_0 \in (0,1)$ and $d_0 \in \mathbb{N}_{\ge 1}$. We say that a family of threshold circuits of size $T(n)$ and depth $d(n)$ is (δ_0, d_0)-*highly uniform *if:*

1. *There exists a $\mathcal{P}$-uniform family of threshold circuits $\{\mathsf{Weight}_{n,i}\}_{n\in\mathbb{N}_{\ge 1},\, i\in[d(n)]}$ of size $T(n)^{\delta_0}$ and depth d_0 such that Weight_n takes $(j,k) \in [T] \times [T]$ as input and outputs $w_{i,j,k}$.*

2. *There exists a $\mathcal{P}$-uniform family of threshold circuits $\{\mathsf{Thr}_{n,i}\}_{n\in\mathbb{N}_{\ge 1},\, i\in[d(n)]}$ of size $T(n)^{\delta_0}$ and depth d_0 such that $\mathsf{Thr}_{n,i}$ takes $j \in [T]$ as input and outputs $\theta_{i,j}$.*[22]

*For convenience, we also say a family of threshold circuits is δ-*highly uniform *if it is $(\delta^2, 1/\delta)$-highly uniform.*

3.2.2 Samplable distributions over circuits, and refuters for them

In this paper we will often consider a distribution over n-input $\mathfrak{C}$-circuits (*i.e.*, a randomized $\mathfrak{C}$ circuits). Since a general distribution may not be described succinctly, we will consider the following two standards to describe randomized $\mathfrak{C}$ circuits:

Definition 3.7 (probabilistic circuits). *A size-s n-input probabilistic $\mathfrak{C}$ circuit C is a $\mathfrak{C}$ circuit that takes two inputs $x \in \{0,1\}^n$ and $r \in \{0,1\}^R$, where $R \le s$ is the number of random coins used by C. Given an input $x \in \{0,1\}^n$, C draws $r \leftarrow \mathcal{U}_R$ and outputs $C(x,r)$.*

Definition 3.8 (samplable distribution over circuits). *Let $\mathfrak{C}, \mathfrak{C}'$ be two circuit classes. We say that a distribution $\mathcal{D}$ over $\mathfrak{C}'$-circuits is $\mathfrak{C}$-*samplable *if there exists a $\mathfrak{C}$-circuit S, which we call a* sampler for $\mathcal{D}$, *that satisfies the following: The circuit S gets random coins as input, prints a description of a $\mathfrak{C}'$-circuit, and the output distribution (over a uniform choice of coins) is exactly $\mathcal{D}$. We say that a family $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ of distributions, where $\mathcal{D}_n$ is a distribution over circuits with n input bits, is* samplable by $\mathfrak{C}$-circuits *if for every $n \in \mathbb{N}$ there is a $\mathfrak{C}$-circuit sampler for $\mathcal{D}_n$. In shorthand, we say that $\{\mathcal{D}_n\}$ is a* probabilistic $(\mathfrak{C} \mapsto \mathfrak{C}')$-circuit family.

[22] More formally, since by definition of threshold circuits we have $0 \le w_{i,j,k}, \theta_{i,j} \le T$, Weight_n and Thr_n both have $\lceil \log T \rceil$ output gates, specifying the binary representation of $w_{i,j,k}$ and $\theta_{i,j}$, respectively.

Loosely speaking, a refuter for f against samplable distributions over circuits gets as input a description of a sampler S, and outputs a string x such that the distribution over circuits fails to compute $f(x)$.

Definition 3.9 (refuter for samplable distributions of circuits). *Let $\mathfrak{C}, \mathfrak{C}'$ be two circuit classes, and let $\tau \in (0,1)$. We say that an algorithm R is a $\mathcal{P}$-computable τ-refuter for f against probabilistic $(\mathfrak{C} \mapsto \mathfrak{C}')$-circuits, if for every probabilistic $(\mathfrak{C} \mapsto \mathfrak{C}')$-circuit family $\{\mathcal{D}_n\}$ and sufficiently large $n \in \mathbb{N}$, when R is given input 1^n and a description of a $\mathfrak{C}$-sampler S_n for $\mathcal{D}_n$, outputs a string $x \in \{0,1\}^n$ such that $\Pr[f(x) = \mathcal{D}_n(x)] \leq \tau$.*

Similarly to Definition 3.4, a compression refuter for f against a distribution over circuits outputs x such that the distribution fails to output a small circuit whose truth-table is $f(x)$.

Definition 3.10 (compression refuter for samplable distributions of circuits). *Let $\mathfrak{C}, \mathfrak{C}'$ be two circuit classes. We say that an algorithm R is a $\mathcal{P}$-computable $(\mathfrak{D}, n^{\varepsilon})$-compression list refuter for f against probabilistic $(\mathfrak{C} \mapsto \mathfrak{C}')$-circuits, if for every probabilistic $(\mathfrak{C} \mapsto \mathfrak{C}')$-circuit family $\{\mathcal{D}_n\}$ and sufficiently large $n \in \mathbb{N}$, when R is given input 1^n and a description of a $\mathfrak{C}$-sampler S_n for $\mathcal{D}_n$, it prints a length-t list $x_1, \ldots, x_t \in \{0,1\}^n$ such that*

$$\text{for some } i \in [t],\ \Pr[\mathcal{D}_n(x_i) \text{ outputs a } \mathfrak{D} \text{ circuit of size } n^{\varepsilon} \text{ whose truth-table is } f(x_i)] < 2/3\,. \quad (1)$$

When we omit the circuit class $\mathfrak{D}$ above, we set it to unrestricted Boolean circuits by default.

Definition 3.11. *Let $\mathfrak{F}$ be a circuit class. We say R is a probabilistic $\mathfrak{F}$-computable τ-refuter for f against probabilistic $(\mathfrak{C} \mapsto \mathfrak{C}')$-circuits, if with probability $1-\tau$, $R(1^n)$ outputs a string $x \in \{0,1\}^n$ such that $\Pr[f(x) = \mathcal{D}_n(x)] \leq \tau$.*

When τ is not specified, we take $\tau = 2/3$ by default (in both definitions of refuters for samplable distributions of circuits).

3.3 Reconstructive PRGs and HSGs

In this section we present known construtions of pseudorandom generators and of (targeted) hitting-set generators. To that end, let us recall the standard notion of a circuit that distinguishes a distribution from the uniform distribution, and of a circuit that avoids a distribution.

Definition 3.12 (Avoiding and Distinguishing). *Let $m, t \in \mathbb{N}$, $D \colon \{0,1\}^m \to \{0,1\}$, and $Z = (z_i)_{i \in [t]}$ be a list of strings from $\{0,1\}^m$. Let $\varepsilon \in (0,1)$. We say that D ε-distinguishes Z, if*

$$\left| \Pr_{r \in \{0,1\}^m}[D(r) = 1] - \Pr_{r \in [t]}[D(z_i) = 1] \right| \geq \varepsilon.$$

We say that D ε-avoids Z, if $\Pr_{r \in \{0,1\}^m}[D(r) = 1] \geq \varepsilon$ and $D(z_i) = 0$ for every $i \in [t]$.

The first PRG is the Nisan-Wigderson [NW94] construction, with flexible parameters and with its reconstruction presented as a distribution over deterministic $\mathcal{TC}^0$ circuits that is samplable by $\mathcal{P}$-uniform probabilistic $\mathcal{TC}^0$ circuits.

Theorem 3.13 (the NW PRG with $\mathcal{TC}^0$ reconstruction). *There are universal constants $c_{\mathtt{NW}} > 1$ and $d_{\mathtt{NW}} \in \mathbb{N}_{\geq 1}$ and deterministic algorithms $G^{\mathtt{NW}}$ and $R^{\mathtt{NW}}$ such that the following holds:*

1. **Generator:** *When given a string* $a \in \{0,1\}^n$ *and* $m \in \mathbb{N}$ *such that* $(\log(n))^{c_{\mathtt{NW}}} \le m \le n^{1/c_{\mathtt{NW}}}$, *the algorithm* $G^{\mathtt{NW}}$ *runs in time* $n^{c_{\mathtt{NW}} \cdot (\log(n)/\log(m))}$, *and prints a list of strings in* $\{0,1\}^m$.

2. **Reconstruction:** *On input* $(1^n, m)$ *such that* $(\log(n))^{c_{\mathtt{NW}}} \le m \le n^{1/c_{\mathtt{NW}}}$, *the algorithm* $R^{\mathtt{NW}}$ *runs in time* $n^{c_{\mathtt{NW}} \cdot (\log(n)/\log(m))}$ *and prints the description of a non-adaptive oracle* $\mathcal{TC}^0_{d_{\mathtt{NW}}}$ *circuit* S *with* $m^{c_{\mathtt{NW}}}$ *gates that maps randomness to a description of a non-adaptive oracle* $\mathcal{TC}^0_{d_{\mathtt{NW}}}$ *circuit* Dec *with* $m^{c_{\mathtt{NW}}}$ *gates. For any oracle* $D\colon \{0,1\}^m \to \{0,1\}$ *that* $1/m$*-distinguishes* $G^{\mathtt{NW}}(a, m)$, *with probability at least* $1 - 2^{-3m}$ *over* Dec *drawn from* S^a, *it holds that*

$$\Pr_{i \in [n]} \left[\mathsf{Dec}^D(i) = a_i \right] \ge 1/2 + m^{-3}.$$

Proof. The algorithm $G^{\mathtt{NW}}$ constructs a combinatorial design $S_1, ..., S_m \subseteq [d]$ with sets of size $|S_i| = \log(n)$ and with pairwise intersections $|S_i \cap S_j| \le 10 \cdot \log(m)$ for distinct $i, j \in [m]$ and $d = 2(\log(n))^2/\log(m)$ (see, e.g., [AB09, Lemma 20.14]). For every $s \in \{0,1\}^d$, the s^{th} output string in the list is $(a_{z\restriction_{S_1}}, ..., a_{z\restriction_{S_m}}) \in \{0,1\}^m$.

Let us describe the oracle circuit S that prints Dec (it will be evident from the description that a polynomial-time algorithm $R^{\mathtt{NW}}$ can print S). For $t = 1, ..., O(m^2)$ in parallel, the circuit S:

1. Randomly chooses $i \in [m]$ and $z \in \{0,1\}^{d-\ell}$ and a bit $\sigma \in \{0,1\}$, and queries a in $\le m \cdot 2^{10 \cdot \log(m)}$ locations according to (i, z, σ) and the design.

2. Randomly chooses $r = O(m^4)$ locations $q_1, ..., q_r \in [n]$, and queries a on these locations.

3. Let Dec^t be a deterministic $\mathcal{AC}^0$ oracle circuit computing the standard reconstruction of [NW94] with the fixed values (i, z, σ) and the fixed design hard-wired into Dec^t. The circuit S prints a deterministic $\mathcal{TC}^0$ oracle circuit Est^t that computes $\nu^t = \Pr_{i \in [r]} \left[(\mathsf{Dec}^t)^D(q_i) = a_{q_i} \right]$. (The circuits Dec^t for $t \in [O(m)]$ will be sub-circuits of Dec.)

Then, the circuit S prints a top gadget for the circuit Dec, which finds t that maximizes ν^t (breaking ties arbitrarily), and on input $i \in [n]$ answers $(\mathsf{Dec}^t)^D(i)$.

Note that both S and Dec are non-adaptive oracle circuits (i.e., S queries a non-adaptively, and Dec queries D non-adaptively) whose depth is bounded by a universal constant $d_{\mathtt{NW}} \in \mathbb{N}$, and whose size is at most $\mathrm{poly}(m) \cdot 2^{10 \cdot \log(m)} \le m^{c_{\mathtt{NW}}}$. By a standard analysis from [NW94], for each t, with probability at least $1/O(m)$ over choice of (i, z, σ) it holds that

$$\mu^t = \Pr_{q \in [n]} \left[(\mathsf{Dec}^t)^D(q) = a_q \right] \ge 1/2 + 1/O(m^2).$$

Hence, with probability $1 - 2^{-\Omega(m)}$, there exists t such that $\mu^t \ge 1/2 + 1/O(m^2)$. Now, conditioned on $|\nu^t - \mu^t| \le 1/m^3$ for all t, which also happens with probability $1 - 2^{-\Omega(m)}$, we have $\Pr_{i \in [n]} \left[(\mathsf{Dec})^D(i) = a_i \right] \ge 1/2 + m^{-3}$. ■

The second PRG is the standard combination of the Nisan-Wigderson [NW94] construction with the error-correcting code of Sudan, Trevisan, and Vadhan [STV01] for hardness amplification. We present it while arguing that the reconstruction is a non-uniform $\mathcal{TC}^0 \circ \mathsf{XOR}$ circuit.

Theorem 3.14 (the STV PRG with $\mathcal{TC}^0 \circ \mathsf{XOR}$ reconstruction)**.** *There are universal constants* $c_{\mathtt{STV}} > 1$ *and* $d_{\mathtt{STV}} \in \mathbb{N}_{\ge 1}$ *such that for every sufficiently small constant* $\gamma \in (0,1)$, *there are deterministic algorithms* $G^{\mathtt{STV}}$ *and* $R^{\mathtt{STV}}$ *that satisfy the following:*

1. **Generator:** *When given a string* $a \in \{0,1\}^n$, G^{STV} *runs in time* $n^{c_{\mathsf{NW}}/\gamma^2}$ *and prints a list of strings in* $\{0,1\}^m$, *where* $m = n^{\gamma}$.

2. **Reconstruction:** $R^{\mathsf{STV}}(1^n)$ *outputs the description of a probabilistic*

$$\left(\mathcal{TC}^0_{d_{\mathsf{STV}}}[n \cdot m^{c_{\mathsf{STV}}}] \mapsto \mathcal{TC}^0_{d_{\mathsf{STV}}} \circ \mathsf{XOR}[m^{c_{\mathsf{STV}}}]\right)$$

oracle circuit $\mathcal{R}_f$, *such that given* $D\colon \{0,1\}^m \to \{0,1\}$ *that* $1/m$*-distinguishes* $G^{\mathsf{STV}}(a)$ *as oracle, we have*

$$\Pr_{R_f \leftarrow \mathcal{R}_f}\left[R_f^D(a) \textit{ outputs a } \mathcal{TC}^0_{d_{\mathsf{STV}}} \textit{ non-adaptive oracle circuit } E \textit{ such that } \mathtt{tt}(E^D) = a\right] \geq 2/3.$$

The fact that the reconstruction can be done with a non-uniform $\mathcal{TC}^0 \circ \mathsf{XOR}$ circuit follows from the original proof, but it is non-standard. We therefore include a proof of this fact in Appendix B.

Next, we present the targeted hitting-set generator of Chen and Tell [CT21]. Specifically, we present the generator while arguing that its reconstruction is a streaming algorithm using bounded space.

Theorem 3.15 (the reconstructive targeted HSG from [CT21] as a streaming algorithm)**.** *There exists a universal constant* $c > 1$ *such that the following holds. Let* $f\colon \{0,1\}^n \to \{0,1\}^n$ *be computable in time* $T(n)$, *let* $\gamma > 0$, *and let* $M\colon \mathbb{N} \to \mathbb{N}$ *such that* $c \cdot \log(T) \leq M \leq T^{\gamma/c}$. *Then, there exists a deterministic algorithm* H_f^{CT} *and a probabilistic oracle machine* R_f^{CT} *that for every* $z \in \{0,1\}^N$ *satisfy the following:*

1. **Generator:** *When given input* z, *the machine* H_f^{CT} *runs in time* $\mathrm{poly}(T(N))$ *and prints a list of strings in* $\{0,1\}^M$.

2. **Reconstruction:** R_f^{CT} *gets input* z, *and can be implemented by an* M^c*-space one-pass streaming algorithm over the input* z *with running time* $M^c \cdot T^{1+\gamma}$. *When* R_f^{CT} *is given oracle access to a function* $D\colon \{0,1\}^M \to \{0,1\}$ *that* $1/M$*-avoids* $H_f^{\mathsf{CT}}(z)$, *with probability at least* $1 - 1/M$ *the machine* R_f^{CT} *outputs an oracle circuit* $C_{f(z)}$ *of size* T^{γ} *such that the truth-table of* $(C_{f(z)})^D$ *is* $f(z)$.

The fact that the reconstruction algorithm of the generator in Theorem 3.15 is a one-pass streaming algorithm was not explicitly stated before, but it follows already from the original construction and proof. For completeness, we explain why this is the case in Appendix A.

3.4 Search-to-decision reduction for randomized algorithms

We will use the following search-to-decision reduction for $pr\mathcal{BPP}$. The reduction constructs an (approximate) solution to a $\mathcal{BPP}$-search problem (as defined in [Gol11]) by repeatedly calling an algorithm for corresponding decision problem. In fact, in the following statement, we consider search problems such that solutions can be verified by circuits from a certain (potentially weak) class $\mathfrak{C}$, and reduce finding (approximate) solutions to such problems to a CAPP-like decision problem for $\mathfrak{C}$. That is:

Theorem 3.16. *Let $\mathfrak{C}$ be a circuit class, and assume that for every $\mu \in (0,1)$ and $c \in \mathbb{N}$ there is a deterministic polynomial-time algorithm that gets as input $C \in \mathfrak{C}$, accepts if $\Pr_r[C(r) = 1] \geq \mu$, and rejects if $\Pr_r[C(r) = 1] \leq \mu - 1/|C|^c$. Then, for every $0 < a < b < 1$, there is a deterministic polynomial-time algorithm that, given a $\mathfrak{C}$ circuit $C\colon \{0,1\}^{\alpha+\beta} \to \{0,1\}$ such that $\Pr_{z \leftarrow \{0,1\}^{\alpha+\beta}}[C(z) = 1] \geq b$, outputs a string x such that $\Pr_{z \leftarrow \{0,1\}^{\beta}}[C(x,z)] \geq a$.*

Proof. The proof is a search-to-decision reduction a-la [Gol11], constructing x bit-by-bit. Starting with x' that is the empty string, we will maintain the invariant that after iteration $i \in [\alpha]$, the updated prefix $x' \in \{0,1\}^i$ will satisfy $\Pr_{r' \in \{0,1\}^{\alpha-i}, z \in \{0,1\}^{\beta}}[C(x'r', z) = 1] \geq b - i/|C|^2$. To do so, in each iteration $i \in [\alpha]$, the algorithm decides whether

$$\Pr_{r' \in \{0,1\}^{\alpha-i}, z \in \{0,1\}^{\beta}}[C(x'0r', z) = 1] \geq b - (i-1)/|C|^2 \tag{3.1}$$

or

$$\Pr_{r' \in \{0,1\}^{\alpha-i}, z \in \{0,1\}^{\beta}}[C(x'0r', z) = 1] \leq b - i/|C|^2 \,, \tag{3.2}$$

by calling the hypothesized deterministic polynomial-time algorithm Est for this problem. If Est accepts, then $x'0$ does not satisfy Eq. (3.2), and we proceed with the i-bit prefix $x'0$; if Est rejects, then $x'0$ does not satisfy Eq. (3.1), and we proceed with the i-bit prefix $x'1$. Since at least one string $x'0$ or $x'1$ satisfies Eq. (3.1), the invariant is maintained after the iteration. After $\alpha \leq |C|$ iterations, we have that $\Pr_{z \in \{0,1\}^{\beta}}[C(x,z) = 1] \geq b - 1/|C| > a$. ■

3.5 Refuting functions with one output bit

Recall that, as stated in Section 1.1, it is straightforward to show that refuters for functions with *a single output bit* implies derandomization. In fact, the proof holds even when the class of refuted algorithms is the weakest possible in terms of dependency on the input:

Claim 3.17. *Assume that there is an $\mathcal{FP}$-refuter for some decision problem $f \in \mathcal{P}$ against the class of probabilistic size-n circuits that are* insensitive to their input *(i.e., their output depends only on the input length). Then, pr$\mathcal{BPP}$ = pr$\mathcal{P}$.*

Proof. Let A be a refuter in $\mathcal{FP}$ for $f \in \mathcal{P}$ against probabilistic circuits that are insensitive to their input; we show how to solve CAPP in deterministic polynomial time. Given a circuit C of size at most n, let D be a probabilistic circuit that ignores its input x, chooses $r \in \{0,1\}^n$ uniformly at random, and outputs $C(r)$; note D also has size n (ignoring the inputs). Given C, our algorithm for CAPP constructs D and runs $A(D)$, printing an x such that $\Pr_r[D(x,r) \neq f(x)] > 1/3$. Since $D(x,r) = C(r)$, we have $\Pr_r[C(r) \neq f(x)] > 1/3$; in other words, we are not in the case that $\Pr_r[C(r) = f(x)] \geq 2/3$. Since $f \in \mathcal{P}$, we can compute $\neg f(x)$ and output it. ■

Note that Open Problem 1 asks to prove a statement as in Claim 3.17 but for arbitrary functions $f \in \mathcal{FP}$, rather than only for decision problem $f \in \mathcal{P}$.

4 A $\mathcal{TC}^0$-locally-encodable and $\mathcal{TC}^0$-locally-approximately-decodable code

Our main goal in this section is to prove the following statement, which asserts that there is an error-correcting code that is locally encodable by $\mathcal{TC}^0$ circuits, and locally approximately decodable by $\mathcal{TC}^0$ circuits. That is:

Proposition 4.1 (a locally encodable and locally approximately decodable code)**.** *There is a universal constant $c_0 > 1$ such that the following holds. For every $\gamma, \nu > 0$ and finite field $\mathbb{F}$ of size $|\mathbb{F}| \le \mathrm{poly}(N)$ there exists $c = c_{\gamma,\nu} > 1$ and a mapping $\mathsf{Enc}\colon \mathbb{F}^N \to \{0,1\}^{\bar{N}}$, where $\bar{N} = N^c$, such that the following holds:*

1. **(Locally encodable.)** *There is a $\mathcal{P}$-uniform family $\{Q_N\}_{N\in\mathbb{N}}$ of threshold circuits of constant depth and size $|Q_N| = N^{c_0\cdot(\gamma+\nu)}$ such that Q_N gets input $i \in [\bar{N}]$ and prints a set $q_1, ..., q_M \in [N]$, where $M = N^\gamma$. Also, there is a $\mathcal{P}$-uniform family $\{E_N\}_{N\in\mathbb{N}}$ of threshold circuits of constant depth and size $|E_N| = N^{c_0\cdot(\gamma+\nu)}$ such that E_N gets input $i \in [\bar{N}]$ and $x_1, ..., x_M \in \mathbb{F}$, and outputs a bit σ such that the following holds: For any $z \in \mathbb{F}^N$ satisfying $z_{q_\ell} = x_\ell$ for all $\ell \in [M]$, the output of E_N is $\sigma = \mathsf{Enc}(z)_i$.*

2. **(Locally approximately decodable.)** *There is a $\mathcal{P}$-uniform family $\{D_N\}_{N\in\mathbb{N}}$ of probabilistic oracle threshold circuits of constant depth and size $|D_N| = N^{c_0\cdot(\gamma+\nu)}$ such that for every $z \in \mathbb{F}^N$ the following holds. The circuit D_N first has a probabilistic preprocessing step, in which it non-adaptively queries z. Now, fix any $O \in \{0,1\}^{\bar{N}}$ satisfying $\Pr_{j\in[\bar{N}]}\left[\mathsf{Enc}(z)_j = O(j)\right] > 1/2 + N^{-\nu}$. Then, with probability at least $1 - o(1)$ over the coins in the preprocessing step, there exists a set $S \subseteq [N]$ of density $|S|/N \ge 1 - N^{-\gamma}$ such that for every $i \in S$,*

$$\Pr\left[(D_N)^O(i) = z_i\right] > 2/3\,,$$

   where the probability is over the random coins of D_N after the preprocessing step.

3. **(Systematic.)** *There is a $\mathcal{P}$-uniform family $\{I_N\}_{N\in\mathbb{N}}$ of non-adaptive oracle threshold circuits of constant depth and size $|I_N| = N^{c_0\cdot(\gamma+\nu)}$ such that I_N gets input $i \in [N]$ and oracle access to an $\bar{N}$-bit string and for every $x \in \mathbb{F}^N$ and every $i \in [N]$ satisfies $I_N(i)^{\mathsf{Enc}(x)} = x_i$.*

At a high-level, the code underlying Proposition 4.1 will be a combination of two different (locally encodable and approximately locally decodable) codes. Loosely speaking, the first code (uniquely) $N^{-\gamma}$-approximately decodes from agreement $1-\delta$ for a small constant δ (i.e., given a codeword that is corrupted on δ of the coordinates, it recovers the unique original message on all but $N^{-\gamma}$ of the coordinates); and the second code δ-approximately decodes from agreement $1/2 + N^{-\nu}$.

We first present the two codes in Sections 4.1 and 4.2, respectively, and then prove Proposition 4.1 in Section 4.3 by combining them in a straightforward way.

4.1 The first code: From distance $N^{-\Omega(1)}$ to distance 0.01

The first code, which we now present, $N^{-\gamma}$-approximately decodes from agreement $1-\delta$.

Proposition 4.2. *There are two universal constants $c_1 > 1$ and $\delta > 0$ such that for every $\gamma > 0$ there exists $\hat{c} = \hat{c}_\gamma > 1$ for which the following holds. Let $\{\mathbb{F}_N\}_{N\in\mathbb{N}}$ be a sequence of finite fields of size $|\mathbb{F}_N| = \mathrm{poly}(N)$. Then, there is a mapping $\mathsf{Enc}_1 \colon (\mathbb{F}_N)^N \to \{0,1\}^{\hat{N}}$, where $\hat{N} = N^{\hat{c}}$, such that the following holds:*

1. **(Locally encodable.)** *There is a $\mathcal{P}$-uniform family $\{Q_N\}_{N\in\mathbb{N}}$ of threshold circuits of constant depth and size $|Q_N| = N^{c_1\cdot\gamma}$ such that Q_N gets input $i \in [\hat{N}]$ and prints a set $q_1, ..., q_M \in [N]$, where $M = N^{c_1\cdot\gamma}$. Also, there is a $\mathcal{P}$-uniform family $\{E_N\}_{N\in\mathbb{N}}$ of threshold circuits of constant depth and size $|E_N| = N^{c_1\cdot\gamma}$ such that E_N gets input $i \in [\hat{N}]$ and $x_1, ..., x_M \in \mathbb{F}_N$, and outputs a bit σ such that the following holds: For any $z \in (\mathbb{F}_N)^N$ satisfying $z_{q_\ell} = x_\ell$ for all $\ell \in [M]$, the output of E_N is $\sigma = \mathsf{Enc}_1(z)_i$.*

2. **(Locally approximately decodable.)** *There is a $\mathcal{P}$-uniform family $\{D_N\}_{N\in\mathbb{N}}$ of probabilistic non-adaptive oracle threshold circuits of constant depth and size $|D_N| = N^{c_1\cdot\gamma}$ such that for every $z \in (\mathbb{F}_N)^N$ the following holds. Let $O \colon \{0,1\}^{\hat{N}} \to \{0,1\}$ such that $\Pr_{j\in[\hat{N}]}\left[\mathsf{Enc}_1(z)_j = O(j)\right] \geq 1-\delta$. Then, there exists a set $S \subseteq [N]$ of density $|S|/N \geq 1 - N^{-\gamma}$ such that for every $i \in S$,*
$$\Pr\left[(D_N)^O(i) = z_i\right] \geq 2/3\,,$$
where the probability is over the random coins of D_N.

3. **(Systematic.)** *There is a $\mathcal{P}$-uniform family $\{I_N\}_{N\in\mathbb{N}}$ of non-adaptive oracle threshold circuits of constant depth and size $|I_N| = N^{c_1\cdot\gamma}$ such that I_N gets input $i \in [N]$ and oracle access to an $\hat{N}$-bit string and for every $z \in \mathbb{F}^N$ and every $i \in [N]$ satisfies $I_N(i)^{\mathsf{Enc}_1(z)} = z_i$.*

At a high level, we will first use the classical expander-based distance-amplification of Alon *et al.* [ABN+92] to increase the distance of the code from $N^{-\gamma}$ to (say) 0.4. Then we will reduce the alphabet to $\{0,1\}$ in a sequence of concatenation steps, where each concatenation step mildly reduces the size of the alphabet while approximately preserving the distance.

Towards presenting the proof, in Sections 4.1.1 and 4.1.2 we construct two building-blocks that will be used repeatedly in the code. Then, in Section 4.1.3 we prove Proposition 4.2. The following auxiliary technical definition will be used in both building-blocks.

Definition 4.3 (nice alphabets)**.** *We say that a sequence $\{\Sigma_M\}_{M\in\mathbb{N}}$ of alphabets of size is* nice *if there are two functions $\Phi = \left\{\Phi_M \colon \Sigma_M \to \{0,1\}^{\lceil\log(|\Sigma_M|)\rceil}\right\}_{M\in\mathbb{N}}$ and $\Phi^{-1} = \left\{\Phi_M^{-1} \colon \{0,1\}^{\lceil\log(|\Sigma_M|)\rceil} \to \Sigma_M\right\}_{M\in\mathbb{N}}$ that are computable in $\mathcal{P}$-uniform $\mathcal{TC}^0$ of size $\mathrm{polylog}(|\Sigma_M|)$ and that satisfy $\Phi_M^{-1}(\Phi_M(x)) = x$ for every $M \in \mathbb{N}$ and $x \in \Sigma_M$.*

4.1.1 Efficient implementation of expander-based distance amplification

The first building-block is an efficient implementation of the expander-based distance amplification of [ABN+92], presented in [GGH+07] (following [GV04]).

Lemma 4.4 (efficient expander-based distance amplification)**.** *There exists $\alpha \in (0,1)$ such that the following holds. Let $\{\Sigma_M\}_{M\in\mathbb{N}}$ be a nice sequence of alphabets, and let $d(M) = M^{O(\gamma)}$ or $d(M) = \mathrm{poly}(|\Sigma_M|)$. Then, there exists $\mathsf{Enc}^{\mathsf{ex}} = \left\{\mathsf{Enc}^{\mathsf{ex}}_M \colon (\Sigma_M)^M \to (\Sigma_M^d)^M\right\}$ such that the following holds.*

1. **(Locally encodable.)** *There is a $\mathcal{P}$-uniform family of $\mathcal{TC}^0$ circuits $\{E_M\}_{M\in\mathbb{N}}$ of size $\mathrm{poly}(d,\log(M))$ such that E_M gets input $i \in [M]$ and prints a set of coordinates $\Gamma_M(i,1),...,\Gamma_M(i,d)$. For every $z \in \Sigma_M^M$, let $\mathsf{Enc}_M^{\mathtt{ex}}(z) \in (\Sigma_M^d)^M$ such that every $i \in [M]$ it holds $\mathsf{Enc}_M^{\mathtt{ex}}(z)_i = (z_{\Gamma_M(i,1)},...,z_{\Gamma_M(i,d)})$.*

2. **(Locally approximately decodable.)** *There is a $\mathcal{P}$-uniform family of non-adaptive oracle $\mathcal{TC}^0$ circuits $\{D_M\}_{M\in\mathbb{N}}$ of size $\mathrm{poly}(d,\log(M),\log(|\Sigma|))$ that satisfies the following. Let $y \in (\Sigma_M^d)^M$ such that there exists $z \in (\Sigma_M)^M$ for which $\Pr_{i\in[M]}\left[\mathsf{Enc}_M^{\mathtt{ex}}(z) = y\right] \geq 0.6$. Then, for all but $d^{-\alpha}$ of the coordinates $i \in [M]$ we have $(D_M)^y(i) = z_i$.*

Proof. Let $\alpha > 0$ be a sufficiently small constant. We consider a family of bipartite graphs $[M] \times [M]$ that are d-biregular and have the following property: for any set $B \subseteq [M]$ of vertices on the right side satisfying $|B| \leq 2M/5$, there are at most $\delta \cdot M$ vertices v on the left side satisfying $|\Gamma(v) \cap B| \geq d/2$, where $\Gamma(v)$ is the list of neighbors of v. As shown in [GGH+07, Claim 4.1] (following [GV04], using powers of the expanders of [GG79]), there exists such a family coupled with a family of $\mathcal{P}$-uniform $\mathcal{AC}^0$ circuits of size $\mathrm{poly}(d,\log(M))$ such that given the name of a vertex v (on either side of the graph), the circuit outputs the list $\Gamma(v)$.

Turning to decoding, consider D_M that gets input $i \in [M]$ and oracle access to $y \in (\Sigma_M^d)^M$ as in the hypothesis. The circuit D_M computes the list $\Gamma(i)$, queries y on each $j \in \Gamma(i)$ to obtain a list of d-tuples, and for each $j \in [d]$ it computes $k_j \in [d]$ such that i is the $(k_j)^{th}$ neighbor of j. The output is the majority vote, over all $j \in [d]$, of the $(k_j)^{th}$ entry in the j^{th} tuple. Note that the majority vote can be computed in $\mathcal{P}$-uniform $\mathcal{TC}^0$ of size $\mathrm{poly}(d,\log(|\Sigma|))$,[23] and hence D_M can be implemented by a $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit of such size. For a standard proof of correctness of this decoder, see e.g. [GGH+07, Proof of Theorem 1.3]. ■

4.1.2 Efficient alphabet reduction

The second building-block, presented next, will be used to reduce the alphabet of a code by an almost exponential factor, while approximately preserving its original constant distance. The building block itself is a mapping of every alphabet symbol to a short sequence of symbols over a smaller alphabet, in a way that supports efficient unique decoding of the original symbol from any sequence that has smll constant distance from the correct encoding.

Lemma 4.5 (efficient alphabet reduction)**.** *Let $\{\Sigma_M\}_{M\in\mathbb{N}}$ be a nice sequence of alphabets. Then, there exists a mapping $\mathsf{Enc}^{\mathtt{al}} = \left\{\mathsf{Enc}_M^{\mathtt{al}} \colon \Sigma_M \to (\Sigma'_M)^{\ell_M}\right\}$, where $|\Sigma'_M| = 2^{\mathrm{polyloglog}(|\Sigma_M|)}$ and $\ell_M = \mathrm{polylog}(|\Sigma_M|)$, such that the following holds.*

1. **(Locally encodable.)** *There is a $\mathcal{P}$-uniform family of $\mathcal{TC}^0$ circuits $\{E_M\}_{M\in\mathbb{N}}$ of size $\mathrm{polyloglog}(|\Sigma|)$ such that E_M gets input $z \in \Sigma_M$ and $i \in [\ell_M]$ and outputs $\mathsf{Enc}_M^{\mathtt{al}}(z)_i$.* [24]

2. **(Locally decodable.)** *There is a $\mathcal{P}$-uniform family of probabilistic non-adaptive oracle $\mathcal{TC}^0$ circuits $\{D_M\}_{M\in\mathbb{N}}$ of size $\mathrm{polyloglog}(|\Sigma|)$ that satisfies the following. Let $y \in (\Sigma'_M)^{\ell_M}$ such that there*

[23]To see this, let $\sigma_1,...,\sigma_d$ be the symbols appearing in the corresponding places in the d tuples. For every $\sigma_j \in \Sigma$, we compute $c_j = |\left\{k : \sigma_k = \sigma_j\right\}|$ in $\mathcal{TC}^0$ of size $\mathrm{poly}(d,\log(|\Sigma|))$. Now we compare the d integers $\left\{c_j\right\}_{j\in[d]}$ in $\mathcal{TC}^0$ of size $\mathrm{poly}(d)$ to find the maximal c_j, and output σ_j.

[24]The uniform circuits receive Σ-symbols and output symbols in binary representation, relying on the efficient bijection between Σ and $\{0,1\}^{\log(|\Sigma|)}$ that exists because Σ is nice.

exists $z \in \Sigma_M$ *for which* $\Pr_{i\in[\ell_M]}\left[\mathsf{Enc}^{\mathtt{al}}_M(z)_i = y_i\right] \geq 0.6$. *Then, for every* $i \in [\log(|\Sigma_M|)]$ *we have that* $\Pr[(D_M)^y(i) = z_i] \geq 2/3$*, where the probability is over the internal coins of* D_M.

3. **(Niceness preserving.)** *The alphabet sequence* $\Sigma' = \{\Sigma'_M\}_{M\in\mathbb{N}}$ *is nice.*

Proof. At a high level, we combine a Reed-Muller encoding over a relatively small field with the expander-based encoding from Lemma 4.4. Towards describing the construction, for simplicity we denote $\Sigma = \Sigma_M$ and $\mathsf{Enc}^{\mathtt{al}} = \mathsf{Enc}^{\mathtt{al}}_M$, etc.

Given $z \in \Sigma$, we identify z with the corresponding vector in $\{0,1\}^{k=\log(|\Sigma|)}$ (using the niceness of the alphabet Σ), and encode it by the low-degree extension view of the Reed-Muller code, with a field $\mathbb{F}'$ of size $|\mathbb{F}'| = 2^{\lceil 2\text{loglog}(k)\rceil} = O(\text{loglog}|\Sigma|)^2$ and interpolation set H of size $|H| = 2^{\lceil \text{loglog}(k)\rceil} = O(\text{loglog}(|\Sigma|))$, and $m = \frac{|H|}{\log(|H|)}$ variables. Note that this yields $z^{(1)} \in (\mathbb{F}')^{|\mathbb{F}|'^m}$. [25]

Now we encode $z^{(1)}$ by the code from Lemma 4.4, instantiated with alphabet $\mathbb{F}'$ and length $\ell = |\mathbb{F}'|^m = \text{polylog}(|\Sigma|)$ and parameter value $d = \text{poly}(|\mathbb{F}|')$, to obtain $z^{(2)} \in ((\mathbb{F}')^d)^\ell$. Note that the alphabet $\mathbb{F}'$ is nice, and hence we can use Lemma 4.4.

Let $\mathsf{Enc}^{\mathtt{al}}(z) = z^{(2)}$, and note that

$$|\mathsf{Enc}^{\mathtt{al}}(z)| = \left((\mathbb{F}')^d\right)^\ell \;;$$

we think of $\mathsf{Enc}^{\mathtt{al}}(z)$ as consisting of ℓ symbols from $\Sigma' = (\mathbb{F}')^d$, and note that

$$|\Sigma'| = (\text{loglog}|\Sigma|)^{\text{polyloglog}(|\Sigma|)} = 2^{\text{polyloglog}(|\Sigma|)}$$

and that Σ' is nice.

Let us first describe the encoding circuit E_M. We map z to $z^{(1)}$ via standard Lagrange interpolation over the field $\mathbb{F}'$ and with $|H| = \text{loglog}(|\Sigma|)$, which can be done by a $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit of size $\text{poly}(m \cdot H, \log(|\mathbb{F}'|)) = \text{polyloglog}(|\Sigma|)$. Then we map $z^{(1)}$ to $z^{(2)}$ via Lemma 4.4, which can also be done by a $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit of size $\text{poly}(d, \log(\ell)) = \text{polyloglog}(|\Sigma|)$.

Turning to decoding of a corrupt codeword $y \in (\Sigma')^\ell$, we will use standard decoding of composed codes. That is, we run the standard unique local decoder for the Reed-Muller code from distance $\Delta = d^{-\alpha} = H/100|\mathbb{F}'|$ (where $\alpha > 0$ is the universal constant from Lemma 4.4), and whenever this decoder accesses a symbol, we answer by running the decoder for the code from Lemma 4.4 and giving it access to y.

Since y is $(1/4)$-close to $\mathsf{Enc}^{\mathtt{al}}(z)$ for some $z \in \Sigma$, it holds that y is $(1/4)$-close to the mapping $z^{(2)}$ of $z^{(1)}$ by $\mathsf{Enc}^{\mathtt{ex}}$. Thus, by Lemma 4.4, there exists $\tilde{z} \in (\mathbb{F}')^{|\mathbb{F}'|^m}$ that agrees with $z^{(1)}$ on all but Δ of the coordinates such that the queries of the local decoder for the Reed-Muller code are answered according to $\tilde{z}$. It follows that for every $i \in [k]$, with high probability, the local decoder for the Reed-Muller code outputs the correct i^{th} symbol in the encoding of z.

As for the complexity of the decoder, first note that its queries are indeed non-adaptive, because the two decoders that it uses are non-adaptive. The unique decoder for the Reed-Muller code can be implemented by $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuits of size $\text{poly}(|H|, \log(|\mathbb{F}'|))$, and the decoder from Lemma 4.4 can be implemented by $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuits of size

$$\text{poly}(d, \log(\ell), \log(|\mathbb{F}'|)) = \text{poly}(|\mathbb{F}'|) = \text{polyloglog}(|\Sigma|)\;.$$

[25] In more detail, let H be the set of vectors in $(\mathbb{F}')^m$ with last $m - \log(|H|)$ coordinates equaling zero. Since $|H|^m \geq k$, we identify each coordinate $i \in [k]$ with a corresponding element $\vec{h}_i \in H$. Given $z \in \{0,1\}^k$, for every $\vec{v} \in (\mathbb{F}')^m$ we define $p(\vec{v}) = \sum_{i\in[k]} \delta_{\vec{h}_i}(\vec{v}) \cdot z_i$. The output is $z^{(1)} = (p(\vec{v}))_{\vec{v}\in(\mathbb{F}')^m}$.

The bound follows by combining both circuits. ■

4.1.3 Proof of Proposition 4.2

For simplicity, denote $\mathbb{F} = \mathbb{F}_N$. Given $z \in \mathbb{F}^N$, we compute $\mathsf{Enc}_1(z)$ in four steps, as follows.

1. Encode z to $z^{(1)} \in (\mathbb{F}^d)^N$ using the code from Lemma 4.4, where $d = N^{O(\gamma)}$.

2. Concatenate $z^{(1)}$ with the code from Lemma 4.5; that is, encode each $(\mathbb{F}^d)$-symbol of $z^{(1)}$ by the code from Lemma 4.5, to obtain $z^{(2)} \in \left(\Sigma^{(2)}\right)^{\ell \cdot N}$, where $|\Sigma^{(2)}| = 2^{\mathrm{polyloglog}(\mathbb{F}^d)} = 2^{\mathrm{polylog}(N, \log(|\mathbb{F}|))}$ and $\ell = \mathrm{polylog}(|\mathbb{F}|^d) = \mathrm{poly}(N, \log(|\mathbb{F}|))$. Denote $N^{(2)} = N \cdot \ell = \mathrm{poly}(N)$.

3. Concatenate $z^{(2)}$ with the code from Lemma 4.5 again, to obtain $z^{(3)} \in \left(\Sigma^{(3)}\right)^{\ell' \cdot N^{(2)}}$, where $|\Sigma^{(3)}| = 2^{\mathrm{polyloglog}(|\Sigma^{(2)}|)} = 2^{\mathrm{polyloglog}(N, \log(|\mathbb{F}|))}$ and $\ell' = \mathrm{polylog}(|\Sigma|^{(2)}) = \mathrm{polylog}(N, \log(|\mathbb{F}|))$. Denote $N^{(3)} = N^{(2)} \cdot \ell' = \mathrm{poly}(N)$.

4. Concatenate $z^{(3)}$ with the good binary code of [STV01], to obtain $z^{(4)} \in \{0,1\}^{\mathrm{poly}(N)}$. We define $\mathsf{Enc}_1(z) = z^{(4)}$ and $\hat{N} = |z^{(4)}| = \mathrm{poly}(N)$.

Local encoding. By the definition of Enc_1, each output bit $i \in [\hat{N}]$ of $\mathsf{Enc}_1(z) = z^{(4)}$ is a function of all the bits encoding of a $\Sigma^{(3)}$-symbol in $z^{(3)}$. In turn, each $\Sigma^{(3)}$-symbol in $z^{(3)}$ is the encoding under Lemma 4.5 of a $\Sigma^{(2)}$-symbol in $z^{(2)}$, and each $\Sigma^{(2)}$-symbol in $z^{(2)}$ is the encoding under Lemma 4.5 of an $\mathbb{F}^d$-symbol in $z^{(1)}$. Finally, each $\mathbb{F}^d$-symbol in $z^{(1)}$ is the concatenation of d symbols in z. It follows that each output bit i of $\mathsf{Enc}_1(z)$ depends on d symbols in z.

We now argue that the mapping of i to the d locations of the symbols in z that affect $\mathsf{Enc}_1(z)_i$ can be computed in $\mathcal{P}$-uniform $\mathcal{TC}^0$ of size $N^{O(\gamma)}$. To see this, note that tracing back i to the relevant location of the symbol in $z^{(3)}$, then further to the relevant location in $z^{(2)}$, and then to the relevant location j_i in $z^{(1)}$ is computable easily from the index i (because the encodings $z^{(1)} \mapsto z^{(2)} \mapsto z^{(3)} \mapsto z^{(4)}$ are concatenations). Given $j_i \in |z^{(1)}|$, we run the circuit E_N from Lemma 4.4 to compute the d locations.

Also, by the constructions of E_N's from Lemmas 4.4 and 4.5, we can compute $\mathsf{Enc}_1(z)_i$ from the values of z in these d locations by a $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit of size $N^{O(\gamma)}$. (The main bottleneck is the encoder from Lemma 4.4, which uses size $\mathrm{poly}(d, \log(N))$ for $d = N^{O(\gamma)}$.)[26]

Local decoding. At a high-level, the decoder D_N implements standard decoding for concatenated codes. Specifically, given $i \in [N]$ and oracle access to O as in our assumption, we:

1. Run the decoder $D_N^{(1)}$ for the code from Lemma 4.4 instantiated with parameter $d = N^{O(\gamma)}$. Whenever it tries to access an $\mathbb{F}^d$-symbol $q_1 \in [N]$, perform Step (2) to obtain the answer.

[26] Note that this does not use the local encoding property of Lemma 4.4; that is, to compute $\mathsf{Enc}_1(z)i$ we compute all the bits of the relevant $\Sigma^{(2)}$-symbols and $\Sigma^{(3)}$-symbols. This causes a size blow-up of $\mathrm{polylog}(N, \log(|\mathbb{F}|))$, which does not affect the complexity of the encoder.

2. For all $j \in [\log(|\mathbb{F}^d|)]$ in parallel, we run the decoder $D_N^{(2)}$ for the code from Lemma 4.5, instantiated with alphabet $\mathbb{F}^d$ and with input j. Whenever the decoder tries to access a $\Sigma^{(2)}$-symbol $q_2 \in [N^{(2)}]$, perform Step (3) to obtain the answer.

3. For all $k \in [\log(|\Sigma^{(2)}|)]$, we run the decoder $D_N^{(3)}$ for the code from Lemma 4.5, instantiated with alphabet $\Sigma^{(2)}$ and with input k. Whenever the decoder tries to access a $\Sigma^{(3)}$-symbol $q_3 \in [N^{(3)}]$, perform Step (4) to obtain the answer.

4. Let $\mathsf{Enc}^{\mathsf{STV}}$ be the encoding of [STV01], and recall that it maps $\log(|\Sigma^{(3)}|)$ bits to $t = \mathrm{polylog}(|\Sigma^{(3)}|) = \mathrm{polyloglog}(N)$ bits. We query O at the t locations corresponding to the encoding of the q^{th} symbol, to obtain an answer $a \in \{0,1\}^t$. Then we enumerate over all messages $m \in \Sigma^{(3)}$, compute $\mathsf{Enc}^{\mathsf{STV}}(m)$ for each m, and output m that maximizes $\Pr_{j \in [t]}[\mathsf{Enc}^{\mathsf{STV}}(m)_j = a_j]$.

Since all the decoders are non-adaptive, the composed decoder is also non-adaptive. Also, the original decoder from Lemma 4.5 is probabilistic and has error probability $1/3$; by naive error-reduction, we can assume that it has error probability $N^{-\omega(1)}$, at the cost of increasing the circuit size by a $\mathrm{polylog}(N)$ factor. (This will not affect our analysis, and it preserves non-adaptivity.)

Let us first bound the complexity of the decoder. It can be implemented by combining four $\mathcal{P}$-uniform probabilistic non-adaptive oracle $\mathcal{TC}^0$ circuits, which yields a circuit of total size

$$\underbrace{\mathrm{poly}(d)}_{D_N^{(1)}} + \underbrace{((1+o(1)) \cdot \log(|\mathbb{F}|^d) \cdot \mathrm{polyloglog}(|\mathbb{F}|^d))}_{D_N^{(2)} \text{ and } D_N^{(3)}} + \underbrace{|\Sigma^{(3)}|}_{\text{decoding } \mathsf{Enc}^{\mathsf{STV}}} \leq N^{O(\gamma)} .$$

The proof of correctness follows a standard proof of correctness for decoding concatenated codes. Specifically, with high probability, all invocations of the decoder from Lemma 4.5 were successful (recall that we reduced its error to $N^{-\omega(1)}$); we condition on this event. Now, for a sufficiently small $\delta > 0$, if the distance of O from $\mathsf{Enc}_1(z)$ is at most δ, then for at most $\sqrt{\delta}$ of the blocks of length t corresponding to encodings of $\Sigma^{(3)}$-symbols in $z^{(3)}$, at most $\sqrt{\delta}$ of the bits in the block are corrupted. Hence, the decoder for $\mathsf{Enc}^{\mathsf{STV}}$ succeeds on at least $\sqrt{\delta}$ of locations $q_3 \in [N^{(3)}]$, which implies that the decoder in Step (3) gets oracle access to a string that is of distance $\sqrt{\delta}$ from $z^{(3)}$. The same logic applies to Step (2), and to Step (1). Relying on Lemma 4.4 and on a sufficiently small choice of $\delta > 0$ (such that $\delta^{1/8} < 2/5$) the decoder maps i to z_i for all but $d^{-\alpha} = N^{-\gamma}$ of the coordinates $i \in [N]$.

Systematic. We are given an index $i \in [N]$, and our goal is to find an output index $i' \in [\hat{N}]$ such that $\mathsf{Enc}_1(z)_{i'} = z_i$ for all z. The main thing that we need to verify is that for the code $\mathsf{Enc}^{\mathsf{ex}} \colon (\Sigma_M)^M \to (\Sigma_M^d)^M$, given an input index $i_0 \in [M]$, we can find $j \in [M]$ and $i' \in [d]$ such that i is the $(i')^h$ symbol in $\mathsf{Enc}^{\mathsf{ex}}(z)_j$. The reason that this suffices is that Enc_1 first encodes $z \mapsto \mathsf{Enc}^{\mathsf{ex}}(z)$, and then performs a sequence of concatenation steps, where each concatenation step encodes each block by a systematic code (i.e., either the combination of the Reed-Muller code, which is systematic, with $\mathsf{Enc}^{\mathsf{ex}}$, which we will now show is indeed systematic; or the code of [STV01], which is systematic).

To verify the claim about $\mathsf{Enc}^{\mathsf{ex}}$, recall that given $i \in [N]$ we can produce the list of neighbors of i (in the degree-d expander graph $[M] \times [M]$ underlying $\mathsf{Enc}^{\mathsf{ex}}$) in $\mathcal{P}$-uniform $\mathcal{AC}^0$ of size $\mathrm{poly}(d, \log(M))$. In our setting we will always have $\mathrm{poly}(d, \log(M)) \leq N^{O(\gamma)}$. Letting j be the first neighbor of i in the list, we can find the index i' of i in the list of neighbors of j by $\mathcal{P}$-uniform $\mathcal{AC}^0$ circuits of size $N^{O(\gamma)}$ (i.e., by computing the list of neighbors of j).

4.2 The second code: From distance 0.01 to distance $1/2 - N^{-\Omega(1)}$

We now present the second code, which $(1-\delta)$-approximately decodes from agreement $1/2 + N^{-\nu}$, for an arbitrarily small constant $\delta > 0$. Note that at such agreement we cannot hope to support unique decoding, and thus this code can be thought of as list-decodable. In the statement below, the code will use a preliminary preprocessing step, to ensure that it can find the right message in the list of possible messages.

Proposition 4.6. *There is a universal constant $c_2 > 1$ such that the following holds. For every $\delta, \nu' > 0$ there exists $c' = c'_{\delta,\nu'} > 1$ and a mapping $\mathsf{Enc}_2 \colon \{0,1\}^{\hat{N}} \to \{0,1\}^{\bar{N}}$, where $\bar{N} = \hat{N}^{c'}$, such that the following holds:*

1. **(Locally encodable.)** *There is a $\mathcal{P}$-uniform family $\{Q_{\hat{N}}\}_{\hat{N}\in\mathbb{N}}$ of $\mathcal{TC}^0$ circuits of size $|Q_{\hat{N}}| = \hat{N}^{c_2\cdot\nu'}$ such that $Q_{\hat{N}}$ gets input $i \in [\bar{N}]$ and prints a set $q_1, ..., q_k \in [\hat{N}]$, where $k \le c_2 \cdot (\nu'/\delta^2) \cdot \log(\hat{N})$. Also, there is a $\mathcal{P}$-uniform family $\{E_{\hat{N}}\}_{\hat{N}\in\mathbb{N}}$ of threshold circuits of constant depth and size $|E_{\hat{N}}| = (\hat{N})^{c_2\cdot\nu'}$ such that $E_{\hat{N}}$ gets input $i \in [\bar{N}]$ and $x_1, ..., x_k \in \mathbb{F}$, and outputs a bit σ such that the following holds: For any $z \in \{0,1\}^{\hat{N}}$ satisfying $z_{q_\ell} = x_\ell$ for all $\ell \in [k]$, the output of $E_{\hat{N}}$ is $\sigma = \mathsf{Enc}_2(z)_i$.*
2. **(Locally approximately decodable.)** *There is a $\mathcal{P}$-uniform family $\{D_{\hat{N}}\}_{\hat{N}\in\mathbb{N}}$ of probabilistic non-adaptive oracle $\mathcal{TC}^0$ circuits of size $|D_{\hat{N}}| = (\hat{N})^{c_2\cdot\nu'}$ such that for every $z \in \{0,1\}^{\hat{N}}$ the following holds. Fix any $O \in \{0,1\}^{\bar{N}}$ satisfying $\Pr_{j\in[\bar{N}]}\left[\mathsf{Enc}_2(z)_j = O_j\right] \ge 1/2 + (\hat{N})^{-\nu'}$. The circuit $D_{\hat{N}}$ first has a probabilistic preprocessing step, in which it non-adaptively queries z. Then, with probability at least $1 - o(1)$ over the coins in the preprocessing step, there exists a set $S \subseteq [\hat{N}]$ of density $|S|/\hat{N} \ge 1 - \delta$ such that $(D_{\hat{N}})^O(i) = z_i$ for every $i \in S$.*
3. **(Systematic.)** *There is a $\mathcal{P}$-uniform family $\{I_{\hat{N}}\}_{N\in\mathbb{N}}$ of non-adaptive oracle threshold circuits of constant depth and size $|I_{\hat{N}}| = (\hat{N})^{c_2\cdot\nu'}$ such that $I_{\hat{N}}$ gets input $i \in [\hat{N}]$ and oracle access to an $\bar{N}$-bit string and for every $z \in \{0,1\}^{\hat{N}}$ and every $i \in [\hat{N}]$ satisfies $I_{\hat{N}}(i)^{\mathsf{Enc}_2(z)} = z_i$.*

Proof. At a high-level, the code is the concatenation of the derandomized direct product code of Impagliazzo and Wigderson [IW97] and of the Hadamard code.

Construction. Let $n = \log(\hat{N})$, let $\varepsilon = (\hat{N})^{-c''\cdot\nu'}$, let $\delta' = \delta/2$, and let $k = (c''/(\delta')^2)\cdot\log(1/\varepsilon)$, for a sufficienty large universal constant $c'' > 1$. Consider the two following algorithms:

1. The $\mathsf{expander\text{-}random\text{-}walk\ sampler}$. Specifically, fix any expander over $\{0,1\}^n$ with constant degree and a sufficiently small (constant) normalized second largest eigenvalue. Lt $\mathsf{Samp} \colon \{0,1\}^{m_1} \to (\{0,1\}^n)^k$ be the function that takes as input a description of a k-length walk on the expander (i.e., an initial n-bit index of a vertex and k indices of edges) and outputs the indices of the k vertices encountered in the walk. Note that $m_1 = n + O(k)$.

2. An efficiently computable combinatorial design $\mathsf{Des}\colon \{0,1\}^{m_2} \times [k] \to \{0,1\}^n$, which takes as input $z \in \{0,1\}^{m_2}$ and the index $i \in [k]$ of a set $S_i \subseteq [n]$ of size $|S_i| = n$, and outputs $z\restriction_{S_i}$. The design has the property that for any $i \neq j$ it holds that $|S_i \cap S_j| \leq (\nu'/2) \cdot n$. We will use designs with $m_2 = O(n/\nu')$ and k as above.[27]

Let $\bar{n} = m_1 + m_2 = O_{\nu',\delta'}(n + \log(1/\varepsilon))$. For any $(y_1, y_2) \in \{0,1\}^{\bar{n}}$ and $i \in [k]$, we define $\mathsf{Loc}(y_1, y_2, i) = \mathsf{Samp}(y_1, i) \oplus \mathsf{Des}(y_2, i) \in \{0,1\}^n$. Then, given $z \in \{0,1\}^{\hat{N}}$, we map it to $z' \in (\{0,1\}^k)^{2^{\bar{n}}}$ such that for any $(y_1, y_2) \in \{0,1\}^{\bar{n}}$ it holds that

$$z'_{y_1,y_2} = \left(z_{\mathsf{Loc}(y_1,y_2,1)}, \ldots, z_{\mathsf{Loc}(y_1,y_2,k)} \right) .$$

The output of $\mathsf{Enc}_2(z)$ is the concatenation of z' with the Hadamard code. Since $k = O_{\delta'}(\log(1/\varepsilon))$, this yields a binary codeword $\mathsf{Enc}(z)$ of length

$$2^{\bar{n}+k} = \hat{N} \cdot (1/\varepsilon)^{c_{\nu'}} = \hat{N}^{c'_{\delta,\nu'}} ,$$

for a sufficiently large $c'_{\delta,\nu'} > 1$.

Local encoding. We prove that there exists a $\mathcal{P}$-uniform family of $\mathcal{TC}0$ circuits of small size for local encoding of the code. We first show that the locations for the derandomized direct product encoding of [IW97] can be computed in uniform $\mathcal{AC}^0$:

Claim 4.6.1. *There is a $\mathcal{P}$-uniform family of $\mathcal{AC}^0$ circuits of size $(\hat{N})^{c''\cdot\nu'}$ that get input $(y_1, y_2) \in \{0,1\}^{\bar{n}}$ and print the set $\{\mathsf{Loc}(y_1, y_2, i)\}_{i \in [k]}$.*

Proof. We instantiate Samp with the Gabber-Galil expander [GG79] of constant degree over $[\hat{N}]$.[28] As was shown in [GV04], there is a $\mathcal{P}$-uniform family of $\mathcal{AC}^0$ circuits of size $\mathrm{poly}(\log(\hat{N}), 2^k) < (\hat{N})^{c''\cdot\nu'}$ that gets as input $i \in [k]$ and the description of a k-length walk (i.e., a starting vertex and a list of indices of edges) and outputs the i^{th} vertex in the walk.[29] In particular, given $y_1 \in \{0,1\}^{m_1}$ and $i \in [k]$, such circuits can output $\mathsf{Samp}(y_1, i)$.

Also, combinatorial designs with parameters as those of Des above are well-known to be computable in time $\mathrm{poly}(k, m_2) \ll \hat{N}$ (see, e.g., [Vad12, Problem 3.2]). We consider a $\mathcal{P}$-uniform family of circuits in which the $\mathcal{P}$-uniform algorithm that constructs the circuit computes such a design and hard-wires it into the circuit; the description of a design is of length $k \cdot n \ll (\hat{N})^{\nu'}$. Given input $y_2 \in \{0,1\}^{m_2}$ and $i \in [n]$, the circuit projects y_2 to the coordinates in the i^{th} set in the design.

By combining the two families of circuits above, we obtain a $\mathcal{P}$-uniform family of $\mathcal{AC}^0$ circuits of size at most $N^{c''\cdot\nu'}$ that, given (y_1, y_2, i), computes $\mathsf{Loc}(y_1, y_2, i)$. (That is, the family computes $\mathsf{Samp}(y_1, i)$ and $\mathsf{Des}(y_2, i)$ in parallel and XORs them.) The claim follows by computing $\mathsf{Loc}(y_1, y_2, i)$ in parallel for all $i \in [k]$. $\square$

[27] There exist designs with a significantly larger number of sets $k = 2^{\Theta(\nu')\cdot n}$, but we will not need such a large k.

[28] A minor technical point is that such expanders are only defined over vertex-set of size that is a square (i.e., N^2 for some $N \in \mathbb{N}$). Since we are considering expanders over the vertex-set $[\bar{N}]$, and we do not mind a quadratic increase in the value of $\hat{N}$ in the previous steps, we may assume without loss of generality that $\hat{N}$ is a square.

[29] In [GV04] this claim is stated only for a specific value of k, but as observed in [GGH+07] the original proof already supports the claim for every k.

For the final encoding of $z \in \{0,1\}^{\hat{N}}$ to $\mathsf{Enc}_2(z) \in \{0,1\}^{\bar{N}}$, note that any output index $\bar{i} \in [\bar{N}]$ can be thought of as a pair (i,j) where $i = (y_1, y_2) \in \{0,1\}^{\bar{n}}$ and $j \in \{0,1\}^k$. The set of coordinates that the $\bar{i}^{th}$ output depends on is the set $S_{\bar{i}} = \{\mathsf{Loc}(y_1,y_2,i)\}_{i \in [k]}$, and the value of the $\bar{i}^{th}$ output is $\oplus_{i\in[k]} j_i \cdot \mathsf{Loc}(y_1,y_2,i)$. By Claim 4.6.1, there is a $\mathcal{P}$-uniform family of $\mathcal{AC}^0$ circuits of size $(\hat{N})^{c''\cdot v'}$ computing the mapping $\bar{i} \mapsto S_{\bar{i}}$, and the output list is of size $|S_{\bar{i}}| = k\cdot$. The final output can be computed by computing a parity over k values, and this can be done in $\mathcal{P}$-uniform $\mathcal{TC}^0$ of size $\mathrm{poly}(k) \ll N^{c''\cdot v'}$.

Systematic. Given $i \in [\hat{N}] \equiv \{0,1\}^n$, the circuit $I_{\hat{N}}$ finds a neighbor j of i in the expander over $\{0,1\}^n$ (that was used for the encoding, in the proof of Claim 4.6.2), and finds the index $\sigma \in [O(1)]$ of the edge that goes from j to i (by trying all $O(1)$ indices in parallel). Let y_1 describe the walk that starts from j, goes along index σ to i in the first step, and proceeds arbitrarily (e.g., walking along index σ for $k-1$ additional steps). Note that $\mathsf{Samp}(y_1, i') = i$. Also let $y_2 = 0^{m_2}$, and note that $\mathsf{Loc}(y_1,y_2,1) = i$. Then, $I_{\hat{N}}$ queries its oracle at the location that corresponds to (y_1,y_2) and to the linear function $f(x_1,...,x_k) = x_1$ (we can assume that this is the first location in the block that corresponds to (y_1,y_2)). As argued in the proof of Claim 4.6.2, by our choice of expander this can be executed by a $\mathcal{AC}^0$ circuit of size $(\hat{N})^{c''\cdot v'}$.

Local approximate decoding. The claimed decodability essentially follows from the classical works of [IW97] and [GL89], yet we spell the argument out in detail to explain why the specific properties that we claim hold.

Let us recall the local decoding algorithm of [IW97], and use the presentation of the construction and proof from [DT23]. For convenience, we denote by $\mathsf{IW}_{\hat{N}}$ the mapping of z to z' defined as above (i.e., $z'_{y_1,y_2} = \left(z_{\mathsf{Loc}(y_1,y_2,1)}, ..., z_{\mathsf{Loc}(y_1,y_2,k)}\right)$). Then, we argue that:

Claim 4.6.2. *There is a $\mathcal{P}$-uniform family of probabilistic non-adaptive oracle $\mathcal{TC}^0$ circuits $\left\{D^{\mathsf{IW}}_{\hat{N}}\right\}_{\hat{N}\in\mathbb{N}}$ of size $(\hat{N})^{c''\cdot v'}$ satisfying the following. Let $w \in \{0,1\}^{\hat{N}}$, and let $\bar{O}\colon \{0,1\}^{\bar{n}} \to \{0,1\}^k$ such that $\Pr_{y_1,y_2\in\{0,1\}^{\bar{n}}}[\bar{O}(y_1,y_2) = \mathsf{IW}_{\hat{N}}(w)_{y_1,y_2}] \geq \varepsilon$. The circuit $D^{\mathsf{IW}}_{\hat{N}}$ first has a probabilistic preprocessing step in which it queries w. Then, with probability at least $1-o(1)$ over the randomness of $D^{\mathsf{IW}}_{\hat{N}}$ in the preprocessing step, there is a set $X \subseteq [\hat{N}]$ of density $|X|/\hat{N} \geq 1-\delta'$ such that for every $x \in X$ it holds that $(D^{\mathsf{IW}}_{\hat{N}})^{\bar{O}}(x) = w_x$ (note that this computational step is deterministic).*

Proof. The uniform circuit is essentially the decoding algorithm of [IW97], as presented in [DT23, Lemma A.2 and the subsequent description]. In the preprocessing step it repeats the following procedure $t = O(n/\varepsilon^2)$ times, in parallel:

> Choose at random a seed $z_1 \in \{0,1\}^{m_1}$ for Samp, and an index $i \in [k]$, and values $\alpha \in \{0,1\}^{m_2-n}$ for the entries of $z_2 \in \{0,1\}^{m_2}$ on coordinates outside S_i. Now query w in parallel on a set of at most $(k-1)\cdot 2^{(v'/2)\cdot n}$ locations, which are determined by (i,α) and by the combinatorial design.[30]

[30]Specifically, in parallel for all $j \in [k] \setminus \{i\}$ do the following. Compute the set $S_i \cap S_j$, iterate in parallel over all choices for $x^{(j)} \in \{0,1\}^{|S_i\cap S_j|}$, and compute the n-bit string x' obtained by placing $x^{(j)}$ in locations $S_i \cap S_j$ and $\alpha\restriction_{S_j}$ in locations $S_j \setminus S_i$. Query w in position $\mathsf{Loc}(z_1,z_2,j) = x' \oplus \mathsf{Samp}(z_1,j)$.

Now, given $x \in [\hat{N}]$, the output is the majority of the outputs of t sub-circuits on x, where each sub-circuit corresponds to one of the experiments in the preprocessing steps (i.e., to a fixed choice of (z_1, i, α)), and performs the following:

1. Compute $x' = x \oplus \mathsf{Samp}(z_1, i)$, complete x' (using α) to $z_2 \in \{0,1\}^{m_2}$, and query $\bar{O}$ on input (z_1, z_2).[31]
2. For each $j \in [k] \setminus \{i\}$, let $c_j \in \{0,1\}$ equal zero iff $\bar{O}(z_1, z_2)_j = w_{\mathsf{Loc}(z_1,z_2,j)}$.
3. For $\ell = \sum_{j \neq i} c_j$, output $\bar{O}(z_1, z_2)_i$ with probability $2^{-\ell}$ and a random bit otherwise.

Since this is precisely the construction of [IW97], its correctness follows from the original proof (see, e.g., [DT23, Proof of Lemma A.2]). In the construction above the second step (after preprocessing) is probabilistic, and the original proof shows that with probability $1 - o(1)$ over coins in the preprocessing phase, there is X of density $1 - \delta'$ such that for every $x \in X$ it holds that $\Pr\left[(D^{\mathsf{IW}}_{\hat{N}})^{\bar{O}}(x) = w_x\right] \geq 0.99$. Using naive error-reduction, we can reduce the error probability from 0.01 to $1/(\hat{N})^2$, and choose random coins for the second step in advance (i.e., in the preprocessing phase). Then, the second step is deterministic, and with probability at least $1 - o(1)$ over the coins in the preprocessing phase, the second step is correct for every $x \in X$.

As for the complexity of the construction, note that the number of queries in the preprocessing step is less than

$$Q = 2^{(\nu'/2) \cdot n} \cdot k \cdot (n/\varepsilon^2) = (\hat{N})^{c'' \cdot \nu'} \ ,$$

and that the size of the circuit is at most

$$O\Big(t \cdot (\hat{N})^{c'' \cdot \nu'} \cdot \mathrm{polylog}(\hat{N})\Big) < \hat{N}^{2c'' \cdot \nu'} \ . \qquad \square$$

Next, we recall the list-decoding algorithm for the Hadamard code from [GL89].

Claim 4.6.3. *There is a $\mathcal{P}$-uniform family $\left\{D^{\mathsf{GL}}_{\hat{N}}\right\}_{\hat{N} \in \mathbb{N}}$ of probabilistic non-adaptive oracle $\mathcal{TC}^0$ circuits of size $(\hat{N})^{c'' \cdot \nu'}$ that satisfies the following. For every $z \in \{0,1\}^{\hat{N}}$ and every $O \in \{0,1\}^{\bar{N}}$ that agrees with $\mathsf{Enc}_2(z)$ on $1/2 + (\hat{N})^{-\nu'}$ of the inputs,*

$$\Pr\left[(D^{\mathsf{GL}}_{\hat{N}})^{O}(x) = \mathsf{IW}_{\hat{N}}(z)_x\right] \geq 2\varepsilon \ ,$$

where the probability is over $x \in \{0,1\}^{\bar{n}}$ and over the random coins of $D^{\mathsf{GL}}_{\hat{N}}$.

We are now ready to construct the final decoder $D_{\hat{N}}$. In the preprocessing step, we repeat the following experiment for $O(1/\varepsilon)$ times, in parallel. For $j = 1, ..., O(1/\varepsilon)$:

1. Run the preprocessing step of $D^{\mathsf{IW}}_{\hat{N}}$.
2. Choose uniformly at random a set of $\ell = O(\log(1/\varepsilon))$ locations $q_1^{(j)}, ..., q_\ell^{(j)} \in [\hat{N}]$, and query w on these locations.

[31] To parse the meaning of this step, note that $\mathsf{Loc}(z_1, z_2, i) = \mathsf{Samp}(z_1, i) \circ x' = x$, so we hope to have $\bar{O}(z_1, z_2)_i = w_x$.

3. Choose in advance fixed random coins $r^{(j)}$ to be used by $D^{\mathsf{GL}}_{\hat{N}}$ and by the second step of $D^{\mathsf{IW}}_{\hat{N}}$.[32]

4. For $i \in [\ell]$, run $D^{\mathsf{IW}}_{\hat{N}}(i)$, and whenever it queries its oracle $\bar{O}$ at location $q' \in \{0,1\}^{\bar{n}}$, answer using $D^{\mathsf{GL}}_{\hat{N}}(q')$. (Both decoders are run using the fixed random coins.) Let $\tilde{w}_i^{(j)}$ be the answer of this procedure.

5. Let $\tilde{v}^{(j)} = \Pr_{i\in[\ell]}\left[\tilde{w}_i^{(j)} = w_i\right]$. If $\tilde{v}^{(j)} \geq 1 - \delta'/2$, consider this experiment successful; otherwise, consider the experiment failed.

Now, let $j^* \in [O(1/\varepsilon)]$ be the index of the first successful experiment (if there was no successful experiment, abort). In the second step, the decoder is given $i \in [\hat{N}]$; it runs $D^{\mathsf{IW}}_{\hat{N}}(i)$ and answers its queries using $D^{\mathsf{GL}}_{\hat{N}}$, where both decoders use the fixed random coins specified by $r^{(j^*)}$.

Observe that the final decoder only non-adaptive oracle queries, and can be implemented by $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuits of size

$$r \cdot \ell \cdot O\Big((\hat{N})^{c''\cdot\nu'} \cdot (\hat{N})^{c''\cdot\nu'}\Big) \leq (\hat{N})^{c_2\cdot\nu'} .$$

As for the correctness of the decoder, note that with probability at least ε over choice of random coins for $D^{\mathsf{GL}}_{\hat{N}}$, there exists a set $X_0 \subseteq \{0,1\}^{\bar{n}}$ of density at least $|X_0|/2^{\bar{n}} \geq \varepsilon$ such that for every $x \in X_0$ it holds that $(D^{\mathsf{GL}}_{\hat{N}})^O(x) = \mathsf{IW}_{\hat{N}}(z)_x$. Whenever this happens, there exists $\bar{O}\colon \{0,1\}^{\bar{n}} \to \{0,1\}^k$ satisfying $\Pr_{y_1,y_2\in\{0,1\}^{\bar{n}}}\left[\bar{O}(y_1,y_2) = \mathsf{IW}_{\hat{N}}(w)_{y_1,y_2}\right] \geq \varepsilon$ such that the queries of $D^{\mathsf{IW}}_{\hat{N}}$ will be answered (by $D^{\mathsf{GL}}_{\hat{N}}$) according to $\bar{O}$. Then, with probability at least $1 - o(1)$ over the coins in the preprocessing step of $D^{\mathsf{IW}}_{\hat{N}}$, there exists a set $X \subseteq [\hat{N}]$ of density at least $1 - \delta'$ such that for every $x \in X$ it holds that $(D^{\mathsf{IW}}_{\hat{N}})^{\bar{O}}(x) = w_x$.

For $j \in [r]$, let $D^{(j)}$ be the decoding procedure that runs $D^{\mathsf{IW}}_{\hat{N}}$ and answers its queries using $D^{\mathsf{GL}}_{\hat{N}}$ where both decoders use the coins specified by $r^{(j)}$. Also let $v_j = \Pr_{x\in[\hat{N}]}\left[(D^{(j)})^O(i) = w_i\right]$. Since we repeat the experiment for $r = O(1/\varepsilon)$ times, with probability $1 - o(1)$ there exists j such that $v_j \geq 1 - \delta'$. Also, with probability at least $1 - o(1)$, for every j it holds that $|v_j - \tilde{v}_j| \leq \delta'/2$. Condition on both events happening. Then, j^* satisfies $v_{j^*} \geq 1 - 2\delta' = 1 - \delta$. By definition, the decoder will answer in the second step according to $D^{(j^*)}$, and hence will answer correctly on at least $1 - \delta$ of the coordinates $x \in [\hat{N}]$. ■

4.3 Proof of Proposition 4.1

Given the two codes in Proposition 4.2 and 4.6, we are now ready to prove Proposition 4.1.

Let γ, ν be the parameters and $\mathbb{F}$ be the finite field. Let $c = c_{\gamma,\nu}$ to be a sufficiently large enough constant to be specified later and c_0 be a sufficiently large universal constant.

[32] We stress that we choose different (independent) random coins for $D^{\mathsf{GL}}_{\hat{N}}$ and for $D^{\mathsf{IW}}_{\hat{N}}$, and denote by $r^{(j)}$ the concatenation of these two fixed choices.

Notation of first code. Let c_1 and δ be the universal constants from Proposition 4.2, and let $\hat{\gamma}$ be a constant to be specified later. We apply Proposition 4.2 with parameters $\hat{\gamma}$ and field $\mathbb{F}$ to obtain the encoding

$$\mathsf{Enc}_1\colon \mathbb{F}^N \to \{0,1\}^{\hat{N}}, \text{ where } \hat{N} = N^{\hat{c}} \text{ and } \hat{c} = \hat{c}_{\hat{\gamma}} \,.$$

We then use $\hat{E}, \hat{Q}, \hat{D}$ to denote the circuits Q, D, N from Proposition 4.2.

Notation of second code. Let c_2 be the universal constant from Proposition 4.6, and let $\bar{\nu}$ be two constants to be specified later. We apply Proposition 4.6 with parameters $\delta, \bar{\nu}$ and to obtain the encoding

$$\mathsf{Enc}_2\colon \{0,1\}^{\hat{N}} \to \{0,1\}^{\bar{N}}, \text{ where } \bar{N} = \hat{N}^{\bar{c}} \text{ and } \bar{c} = \bar{c}_{\delta,\bar{\nu}} \,.$$

We then use $\bar{E}, \bar{Q}, \bar{D}$ to denote the circuits Q, D, N from Proposition 4.6.

The mapping Enc. With the notation set up, we now define the encoding map $\mathsf{Enc}\colon \mathbb{F}^N \to \{0,1\}^{\bar{N}}$ as

$$\mathsf{Enc}(x) = \mathsf{Enc}_2(\mathsf{Enc}_1(x)), \text{ where } x \in \mathbb{F}^N \,.$$

We also let $\hat{x} = \mathsf{Enc}_1(x)$ and $\bar{x} = \mathsf{Enc}_2(\hat{x})$. That is, we have the following

$$\mathsf{Enc}\colon x \in \mathbb{F}^N \xrightarrow{\mathsf{Enc}_1} \hat{x} \in \{0,1\}^{\hat{N}} \xrightarrow{\mathsf{Enc}_2} \bar{x} \in \{0,1\}^{\bar{N}}.$$

In particular, we now set $c = \hat{c} \cdot \bar{c}$ so that $\bar{N} = N^c$.

The construction of Q_N. Now we are ready to define Q_N, which is going to be a natural composition of $\bar{Q}_{\hat{N}}$ and $\hat{Q}_N$. Formally, Q_N works as follows:

1. Given input $i \in [\bar{N}]$, run $\bar{Q}_{\hat{N}}(i)$ to obtain $q_1, \ldots, q_{\bar{M}} \in [\hat{N}]$ where $\bar{M} = c_2 \cdot (\bar{\nu}/\delta^2) \cdot \log(\hat{N})$.[33]
2. For every $j \in [\bar{M}]$, run $\hat{Q}_N(q_j)$ to obtain $q_{j,1}, \ldots, q_{j,\hat{M}} \in [N]$ where $\hat{M} = N^{c_1 \cdot \hat{\gamma}}$.
3. Output all of $q_{j,\ell}$ for $j \in [\bar{M}]$ and $\ell \in [\hat{M}]$.

Now, note that Q_N has $M = \bar{M} \cdot \hat{M} \leq N^{2 \cdot c_1 \cdot \hat{\gamma}}$ outputs in $[N]$. We now set $\hat{\gamma} = \gamma/(2 \cdot c_1)$ and $\bar{\nu} = \nu/\hat{c}$. We then have $M \leq N^{\gamma}$ as desired.

Also, we have $|\bar{Q}_{\hat{N}}| \leq \hat{N}^{c_2 \cdot \bar{\nu}}$ and $|\hat{Q}_N| \leq N^{c_1 \cdot \hat{\gamma}}$, it follows their composition Q_N is a $\mathcal{TC}^0$ circuit of size

$$O\left(\hat{N}^{c_2 \cdot \bar{\nu}} + \bar{M} \cdot N^{c_1 \cdot \hat{\gamma}}\right) \leq O\left(N^{c_2 \cdot \bar{\nu} \cdot \hat{c}} + N^{2 \cdot c_1 \cdot \hat{\gamma}}\right) \leq N^{c_0 \cdot (\gamma + \nu)},$$

for a sufficiently large constant c_0, by our choice of $\bar{\nu}$ and $\hat{\gamma}$.

[33] For simplicity, we add some dummy queries to make the number of queries exactly $c_2 \cdot (\bar{\nu}/\delta^2) \cdot \log(\hat{N})$.

The construction of E_N. After defining Q_N, we are now ready to define E_N in the natural way.

1. Given input $i \in [\bar{N}]$ and $x_{j,\ell} \in \mathbb{F}$ for $(j,\ell) \in [\bar{M}] \times [\hat{M}]$, $\bar{Q}_{\hat{N}}(i)$ to obtain $q_1, \ldots, q_{\bar{M}} \in [\hat{N}]$.
2. For every $j \in [\bar{M}]$, run $\hat{E}_N$ with input $q_j \in [\hat{N}]$ and list $q_{j,1}, \ldots, q_{j,\hat{M}}$ to obtain $\sigma_j \in \{0,1\}$.
3. Run $\bar{E}_{\hat{N}}$ with input i and list $\sigma_1, \ldots, \sigma_{\bar{M}}$ to obtain the output σ.

Similarly to the case of Q_N, we can implement E_N by a $\mathcal{TC}^0$ circuit of size $N^{c_0 \cdot (\gamma+\nu)}$. Moreover, the desired properties of Q_N and E_N follows immediately from the properties of $\hat{Q}, \hat{E}, \bar{Q}, \bar{E}$ from Proposition 4.2 and Proposition 4.6.

The construction of D_N. Again, D_N is given by the natural composition of $\bar{D}_{\hat{N}}$ and $\hat{D}_N$. Formally, it works as follows:

1. Given an oracle $O\colon \{0,1\}^{\bar{N}} \to \{0,1\}$ such that
$$\Pr_{j \in [\bar{N}]} \left[\bar{x}_j = O(j)\right] > 1/2 + \left(\hat{N}\right)^{-\bar{\nu}} .$$
2. (**Preprocessing phase.**) Run the preprocessing phase of $\bar{D}_{\hat{N}}$ to obtain non-adaptive queries $q_1, \ldots, q_t \in \left[\hat{N}\right]$ to $\hat{x}$, where $t \le |\bar{D}_{\hat{N}}|$, run $\hat{Q}_N$ to convert these into non-adaptive queries $\{q_{j,\ell}\}_{j\in[t],\ell\in[\hat{M}]}$ to x, and run $\hat{E}_N$ to convert the answers of the new queries to answers of the original queries.
3. Run the main phase of $\bar{D}_{\hat{N}}$ $\Theta(\log \hat{N})$ times with independent randomness, taking a majority, and fixing the randomness to obtain a deterministic oracle circuit $W\colon \{0,1\}^{\hat{N}} \to \{0,1\}$ such that the following
$$\Pr_{j \in \left[\hat{N}\right]} \left[\hat{x}_j = W^O(j)\right] \ge 2/3$$
happens with $1 - o(1)$ probability over all randomness above.[34]
4. (**Main phase.**) Given input $i \in [N]$, output
$$\left(\hat{D}_N\right)^{W^O}(i).$$

Now, D_N can be implemented by a probabilistic $\mathcal{TC}^0$ circuit, and its size can be bounded as follows
$$\begin{aligned}
|D_N| &\le O\left(|\bar{D}_{\hat{N}}| \cdot \log \hat{N} + |\bar{D}_{\hat{N}}| \cdot N^{c_1 \cdot \hat{\gamma}}\right) \\
&\le O\left(\left(\hat{N}\right)^{c_2 \cdot \bar{\nu}} \cdot N^{c_1 \cdot \hat{\gamma}}\right) && (\log \hat{N} \le N^{c_1 \cdot \hat{\gamma}}) \\
&\le O\left(N^{\hat{c} \cdot c_2 \cdot \bar{\nu} + c_1 \cdot \hat{\gamma}}\right) . && (\hat{N} = N^{\hat{c}})
\end{aligned}$$

Note that the required approximation is $1/2 + N^{-\nu}$. Recall that $\bar{\nu} = \nu/\hat{c}$, we have $\left(\hat{N}\right)^{-\bar{\nu}} = N^{-\hat{c}\cdot\bar{\nu}} = N^{-\nu}$. And the size of D_N can be bounded by $N^{c_0 \cdot (\gamma+\nu)}$ by our choice of $\bar{\nu}$ and $\hat{\gamma}$, and setting c_0 to be large enough. The correctness of D_N follows directly from Proposition 4.2 and Proposition 4.6, which completes the proof.

[34]Note that both phases of $\bar{D}$ are considered as the preprocessing phase of D_N. The execution of $\hat{D}$ below is considered as the main phase of D_N.

Systematic. Finally, we note that since both Enc_1 and Enc_2 are systematic, their composed code Enc is systematic as well. This completes the proof.

5 Improved Chen-Tell hitting set generator with $\mathcal{TC}^0$ reconstruction

The goal of this section is to prove the following result, which is an improved version of the targeted hitting-set generator of [CT21]:

Theorem 5.1 (Reconstructive targeted HSG for highly uniform $\mathcal{TC}^0$ circuits)**.** *Let $c \in \mathbb{N}_{\geq 1}$ be a sufficiently large universal constant. For every $\gamma \in (0,1)$ and $d \in \mathbb{N}_{\geq 1}$ there exist $d_1 \in \mathbb{N}_{\geq 1}$ and $\delta \in (0,1)$ such that the following holds. Let $T, M, m \colon \mathbb{N} \to \mathbb{N}$ be such that $M \leq T^{\gamma/c}$. Let $f \colon \{0,1\}^n \to \{0,1\}^{m(n)}$ be computable by a family of δ-highly uniform threshold circuits of depth d and T size. Then, there exist deterministic algorithms $H_f^{\mathsf{CT\text{-}TC}^0}$ and $R_f^{\mathsf{CT\text{-}TC}^0}$ that for every $z \in \{0,1\}^n$ the following hold:*

1. **Generator:** *When given input z, the machine $H_f^{\mathsf{CT\text{-}TC}^0}$ runs in time $\mathrm{poly}(T)$ and prints a set of strings in $\{0,1\}^M$.*
2. **Compression Reconstruction:** *$R_f^{\mathsf{CT\text{-}TC}^0}(1^n)$ outputs the description of a probabilistic*
$$\left(\mathcal{TC}^0_{d_1}[n \cdot T^\gamma] \mapsto \mathcal{TC}^0_{d_1} \circ \mathsf{SUM}[T^\gamma]\right)$$
oracle circuit $\mathcal{R}_f$, such that given $D \colon \{0,1\}^M \to \{0,1\}$ that $1/M$-avoids $H_f^{\mathsf{CT\text{-}TC}^0}(z)$ as oracle, we have
$$\Pr_{R_f \leftarrow \mathcal{R}_f}\left[R_f^D(z) \text{ outputs a } \mathcal{TC}^0_{d_1} \text{ oracle circuit } E \text{ such that } \mathsf{tt}(E^D) = f(z)\right] \geq 2/3.$$

The proof of Theorem 5.1 relies on the $\mathcal{TC}^0$-locally-encodable and $\mathcal{TC}^0$-locally-approximately-decodable code from Section 4. In Section 5.1 we present the construction of a bootstrapping system for highly uniform $\mathcal{TC}^0$ circuit, whose high-level description was given in Section 2.4.1, and in Section 5.2 we present the proof of Theorem 5.1.

5.1 Efficient polynomial decompositions of highly uniform threshold circuits

Towards constructing the bootstrapping system, let us now define an intermediary object called a $\mathsf{polynomial\ decomposition}$ of a circuit. This object, following the ideas of [GKR15], was defined in [CT21] for general (logspace-uniform) circuits, and we present another definition that is suitable for $\mathcal{TC}^0$ circuits.

Definition 5.2 (polynomial decomposition of a threshold circuit)**.** *Let C be a threshold circuit that has n input bits, size T, and depth d. For every $x \in \{0,1\}^n$, we call a collection of polynomials a* $\mathsf{polynomial\ decomposition}$ of $C(x)$ *if it meets the following specifications.*

1. **(Notation.)** *For any $i \in [d]$ and $j \in [T]$, denote by $g_{i,j}$ the j^{th} gate in the i^{th} layer, and denote the function that $g_{i,j}$ computes by*
$$g_{i,j}(x) = \mathbf{1}\left[\sum_{k \in [T]} w_{i,j,k} \cdot g_{i-1,k}(x) > \theta_{i,j}\right] ,$$

where $\theta_{i,j} \in \mathbb{Z}$ and $w_{i,j,k} \in \mathbb{Z}$ for all $k \in [T]$.

2. **(Arithmetic setting.)** *For some prime $5 \cdot T^2 < p \leq 10 \cdot T^2$, the polynomials are defined over the prime field $\mathbb{F} = \mathbb{F}_p$. For some integer $h \leq p$, let $H = [h] \subseteq \mathbb{F}$, let m be the minimal integer such that $h^m \geq T$. Let $\xi \colon [T] \to H^m$ be an injection and $\xi^{-1} \colon H^m \to [T] \cup \{\perp\}$ be its inverse.*[35]

3. **(Circuit-structure polynomial.)** *For each $i \in [d]$, let $\Phi_i \colon H^{2m} \to \{-T, ..., T\}$ be the following function. On input $(\vec{u}, \vec{v}) \in H^m \times H^m$, we interpret the pair as $(j, k) \in [T] \times [T]$, and output $w_{i,j,k}$.*[36] *The polynomial $\hat{\Phi}_i \colon \mathbb{F}^{2m} \to \mathbb{F}$ can be any extension of Φ_i.*

4. **(Input polynomial.)** *Let $\alpha_0 \colon H^m \to \{0,1\}$ represent the string $x0^{h^m - n}$, and let $\hat{\alpha}_0 \colon \mathbb{F}^m \to \mathbb{F}$ be defined by*

$$\hat{\alpha}_0(\vec{u}) = \sum_{i \in [n],\ z = \xi(i)} \delta_{\vec{z}}(\vec{u}) \cdot \alpha_0(\vec{z}) \ ,$$

where $\delta_{\vec{z}}$ is Kronecker's delta function (i.e., $\delta_{\vec{z}}(\vec{u}) = \prod_{j \in [m]} \prod_{a \in H \setminus \{z_j\}} \frac{u_j - a}{z_j - a}$).

5. **(Layer polynomials.)** *For each $i \in [d]$, let $\alpha_i \colon H^m \to \{0,1\}$ represent the values of the gates at the i^{th} layer of C in the computation of $C(x)$ (with zeroes in locations that do not index valid gates).*[37] *Also let*

$$\hat{\alpha}_i(\vec{u}) = \sum_{\vec{v} \in H^m} \hat{\Phi}_i(\vec{u}, \vec{v}) \cdot \hat{\alpha}_{i-1}(\vec{v}) \ .$$

6. **(Sumcheck polynomials.)** *For each $i \in [d]$, let $\hat{\alpha}_{i,0} \colon \mathbb{F}^{2m} \to \mathbb{F}$ be the polynomial*

$$\hat{\alpha}_{i,0}(\vec{u}, \sigma_1, ..., \sigma_m) = \hat{\Phi}_i(\vec{u}, \sigma_{1,...,m}) \cdot \hat{\alpha}_{i-1}(\sigma_{1,...,m}) \ ,$$

and for every $j \in [m-1]$, let $\hat{\alpha}_{i,j} \colon \mathbb{F}^{2m-j} \to \mathbb{F}$ be the polynomial

$$\hat{\alpha}_{i,j}(\vec{u}, \sigma_1, ..., \sigma_{m-j}) = \sum_{\sigma_{m-j+1}, ..., \sigma_m \in H} \hat{\Phi}_i(\vec{u}, \sigma_{1,...,m}) \cdot \hat{\alpha}_{i-1}(\sigma_{1,...,m}) \ ,$$

where $\sigma_{k,...,k+r} = \sigma_k, \sigma_{k+1}, ..., \sigma_{k+r}$. We also denote $\hat{\alpha}_{i,m} \equiv \hat{\alpha}_i$.

Next, we argue that every highly uniform family of $\mathcal{TC}^0$ circuits has a very efficient polynomial decomposition. The crux of the proof is arithmetizing the "weights function" Φ appropriately, relying on a suitable arithmetization of the (small) uniform circuits for Φ (which exist because the family is highly uniform).

Proposition 5.3 (efficient polynomial decompositions of highly uniform threshold circuits)**.** *There exists a universal constant $c \in \mathbb{N}$ such that the following holds. Let $\delta > 0$ be a sufficiently small constant, and let $\{C_n\}_{n \in \mathbb{N}}$ be a δ-highly uniform family of circuits of size $T(n)$ and depth $d(n)$. Then, for every $x \in \{0,1\}^n$ there exists a polynomial decomposition of $C_n(x)$ satisfying:*

[35] $\xi^{-1}(\vec{u}) = \perp$ if $\vec{u}$ is not in the range of ξ. We always use ξ to encode an index i as an element from H^m. We will pick an ξ such that ξ^{-1} is also easy to compute, and for simplicity we ignore the complexity of computing ξ and ξ^{-1} since it is negligible; we only need them to be computable in $\mathcal{TC}^0$.

[36] If $\vec{u}$ or $\vec{v}$ represents an integer larger than T, then $\Phi_i(\vec{u}, \vec{v}) = 0$.

[37] Formally, for every $\vec{u} \in H^m$, $\alpha_i(\vec{u})$ equals $g_{i,\xi^{-1}(\vec{u})}$ if $\xi^{-1}(\vec{u}) \neq \perp$, and 0 otherwise.

1. **(Arithmetic setting.)** *The polynomials are defined over $\mathbb{F}_p$, where p is the smallest prime in the interval $\left[5 \cdot T^2+1,\ 10 \cdot T^2\right]$. Let $H=[h] \subseteq \mathbb{F}$, where h is the smallest power of two of magnitude at least $\left(T^2\right)^{\delta/6}$, and let m be the minimal integer such that $h^m \geq 2T$. Moreover, all polynomials in the polynomial decomposition have total degree at most $T^{c \cdot \delta}$.*

2. **(Faithful representation.)** *For every $i \in [d(n)]$ and $\vec{u} \in H^m$ representing a gate in the i^{th} layer, the value of the gate in $C_n(x)$ is 1 if and only if $\hat{\alpha}_i(\vec{u}) \geq \theta_{i,\vec{u}}$.*

3. **(Base case.)** *There is $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit of size $n \cdot h^c$ that given $\vec{u} \in \mathbb{F}^m$, outputs the description of a* SUM *gate $C_{\vec{u}}$ such that $C_{\vec{u}}(x)=\hat{\alpha}_0(\vec{u})$.*

4. **(Downward self-reducibility.)** *There are two $\mathcal{P}$-uniform non-adaptive oracle threshold circuits of size h^c and constant depth that solve each of the following tasks, respectively:*
   - *(a) Given input $i \in [d]$ and $(\vec{u}, \sigma_1, ..., \sigma_m) \in \mathbb{F}^{2m}$ and oracle access to $\hat{\alpha}_{i-1}$, output $\hat{\alpha}_{i,0}(\vec{u}, \sigma_1, ..., \sigma_m)$.*
   - *(b) Given input $(i,j) \in [d] \times [m]$ and $(\vec{u}, \sigma_1, ..., \sigma_{m-j}) \in \mathbb{F}^{2m-j}$ and oracle access to $\hat{\alpha}_{i,j-1}$, output $\hat{\alpha}_{i,j}(\vec{u}, \sigma_1, ..., \sigma_{m-j})$.*

Proof. To specify the polynomial decomposition according to Definition 5.2, it suffices to specify the circuit structure polynomials $\hat{\Phi}_i$.

Construction of polynomials $\hat{\Phi}_i$. Since $\{C_n\}_{n \in \mathbb{N}}$ is δ-highly uniform, we have:

1. There exists a $\mathcal{P}$-uniform family of threshold circuits $\{\mathsf{Weight}_{n,i}\}_{n \in \mathbb{N}_{\geq 1},\ i \in [d(n)]}$ of size $T(n)^{\delta^2}$ and depth $1/\delta$ such that $\mathsf{Weight}_{n,i}$ takes $(j,k) \in [T] \times [T]$ as input and outputs $w_{i,j,k}$.

2. There exists a $\mathcal{P}$-uniform family of threshold circuits $\{\mathsf{Thr}_{n,i}\}_{n \in \mathbb{N}_{\geq 1},\ i \in [d(n)]}$ of size $T(n)^{\delta^2}$ and depth $1/\delta$ such that $\mathsf{Thr}_{n,i}$ takes $j \in [T]$ as input and outputs $\theta_{i,j}$.

First, by composing with ξ^{-1} from Definition 5.2, we can convert $\mathsf{Weight}_{n,i}$ into a circuit $D_{n,i} \colon H^{2m} \to \mathbb{F}_p$ such that

$$D_{n,i}(\vec{u}, \vec{v}) = \begin{cases} \mathsf{Weight}_{n,i}(\xi^{-1}(u), \xi^{-1}(v)) \bmod p & \xi^{-1}(u) \neq \perp \text{ and } \xi^{-1}(v) \neq \perp\ , \\ 0 & \text{otherwise}\ . \end{cases}$$

In above, for any $z \in \mathbb{N}$, we use $z \bmod p$ to denote its unique conjugate number in $\mathbb{F}_p$. In more detail, $D_{n,i}$ takes $2m$ blocks of length-$\lceil \log h \rceil$ Boolean strings as input, interpret each of them as an integer in H (if any of the strings does not encode a valid integer in H, $D_{n,i}$ outputs 0 immediately) to obtain a pair $(\vec{u}, \vec{v}) \in H^m \times H^m$, and outputs $\mathsf{Weight}_{n,i}(\xi^{-1}(u), \xi^{-1}(v))$ if $\xi^{-1}(u) \neq \perp$ and $\xi^{-1}(v) \neq \perp$ and 0 otherwise. It is easy to see that $D_{n,i}$ can be implemented by a $T^{O(\delta^2)}$-size, $(1/\delta + O(1))$-depth $\mathcal{TC}^0$ circuit.

Now, we can see that $D_{n,i}$ computes Φ_i as per Definition 5.2. To obtain the desired arithmetization $\hat{\Phi}_i \colon \mathbb{F}^{2m} \to \mathbb{F}$, we compute a degree-$h$ polynomial $Q \colon \mathbb{F} \to \mathbb{F}^{\lceil \log h \rceil}$ by interpolation such that for every $u \in [H]$, we have that $Q(u)$ equals the binary representation of u as an integer.

For the next step we will need the following lemma, which allows us to transform the $D_{n,i}$'s to circuits that compute polynomials of bounded degree.

Lemma 5.3.1. *There is a universal constant $c \in \mathbb{N}_{\geq 1}$ and a polynomial-time algorithm that takes a prime $5 \cdot t^2 < p \leq 2^t$ together with the description of a t-size d-depth n-input $\mathcal{TC}^0$ circuit $C\colon \{0,1\}^n \to \mathbb{F}_p$ as input, and outputs another t^c-size $(c \cdot d)$-depth $\mathcal{TC}^0$ circuit $C'\colon \mathbb{F}_p^n \to \mathbb{F}_p$ such that the following hold:*

- *C' computes a degree-$t^{c \cdot d}$ polynomial over $\mathbb{F}_p$.*
- *For every $z \in \{0,1\}^n$, we have $C(z) = C'(z)$.*

Proof. Let $m = \lceil \log p \rceil$. Note that C can be decomposed into m Boolean output circuits $C_1, \dots, C_m$, such that $C_i(z)$ outputs the i-th bit of the binary representation of $C(z)$. Note that in the field $\mathbb{F}_p$, a negative number $-z$ for $z \in [p-1]$ equals $p - z$.

For each $i \in [m]$, we will construct a low-degree polynomial Φ_i such that $C_i(z) = \Phi_i(z)$ for every $z \in \{0,1\}^n$, and show Φ_i can be computed by another $\mathcal{TC}^0$ circuit C_i'. Then, we will combine all the C_i' into a a single circuit C', and all the Φ_i into a single polynomial Φ.

Fix $i \in [m]$, for every gate $G\colon \{0,1\}^v \to \{0,1\}$ in C_i (here $v \leq t$), we have

$$G(y_1, \dots, y_v) := \mathbf{1}\left[\sum_{i \in [v]} w_i \cdot y_i \geq \theta\right],$$

where $w_i, \theta \in \{-t, -t+1, \dots, t\}$ for every $i \in [v]$. By standard interpolation, we can interpolate a degree-$2t^2$ polynomial $p_G\colon \mathbb{F}_p \to \mathbb{F}_p$ such that $p_G(z) = \mathbf{1}[z \geq \theta]$ for every $z \in \{-t^2, -t^2 + 1, \dots, t^2\}$.[38] Moreover, p_G can be computed in $\mathcal{TC}^0$ of size $\mathrm{poly}(t) \cdot \mathrm{polylog}(p)$ [HAB02a]. Next, we define $P_G\colon \mathbb{F}_p^v \to \mathbb{F}_p$ as $P_G(y_1, \dots, y_v) = p_G(\sum_{i \in [v]} w_i \cdot y_i)$. Note that P_G has degree $2 \cdot t^2$ as well, and can also be computed in $\mathcal{TC}^0$.

Now, we replace every gate G in C_i by a degree-$(2 \cdot t^2)$ polynomial P_G over $\mathbb{F}_p$ to obtain a polynomial Φ_i. Since C_i has depth d, we know that Φ_i has degree at most $(2t)^{2d}$. Moreover, by the description above and our assumption on p, Φ_i can be computed by a $\mathrm{poly}(t)$-size $O(d)$-depth $\mathcal{TC}^0$ circuit C_i' that can be constructed from C_i in polynomial time.

Finally, we set $\Phi(z) = \sum_{i=1}^{m} 2^{i-1} \cdot \Phi_i(z)$ for every $z \in \mathbb{F}_p^m$. Note that Φ has degree $(2t)^{2d}$ and Φ can be computed by a $\mathrm{poly}(t)$-size $O(d)$-depth $\mathcal{TC}^0$ circuit C' that can be constructed from C in polynomial time. $\square$

Let $t = T^{O(\delta^2)}$. Noting that $p > 5 \cdot T^2 > 5t^2$, we also apply Lemma 5.3.1 to $D_{n,i}$ to obtain a circuit $D_{n,i}'$ of size $T^{O(\delta^2)}$ and depth $O(1/\delta)$ that computes a degree-$T^{O(\delta)}$ polynomial from $\mathbb{F}^{2m\lceil \log h \rceil}$ to $\mathbb{F}$ that agrees with $D_{n,i}$ on all Boolean inputs. Finally, we define

$$\hat{\Phi}_{n,i}(v_1, \dots, v_{2m}) = D_{n,i}'(Q(v_1), \dots, Q(v_{2m})).$$

From the above discussion, $\hat{\Phi}_{n,i}$ can be computed by a $\mathcal{P}$-uniform threshold circuit family of size $T^{O(\delta^2)}$ and depth $O(1/\delta)$, and it has degree at most $T^{O(\delta)}$, given our choice of h. Most importantly, it is an extension of Φ_i defined in Definition 5.2 (by identifying negative numbers as their conjugates in $\mathbb{F}_p$).

[38] Here $\{-t^2, -t^2+1, \dots, t^2\}$ denotes $\{p - t^2, p - t^2 + 1, \dots, p-1, 0, 1, \dots, t^2\}$.

Verification of properties. After specifying the extension, we immediately obtain the polynomials $\hat{\alpha}_i$ and $\hat{\alpha}_{i,j}$ for $i \in [d]$ and $j \in [m]$. Also, $\hat{\alpha}_0$ is defined as in Definition 5.2. From their definitions and our choice of h, the arithmetic setting, the faithful representation, and the downward self-reducibility follow immediately.

To verify the base case, given $\vec{u} \in \mathbb{F}^m$, we need to output the weights of the SUM gate

$$\left(\delta_{\xi(1)}(\vec{u}), \ldots, \delta_{\xi(n)}(\vec{u})\right).$$

The base case follows from the observation that each entry can be computed by a $\mathrm{poly}(h)$-size $\mathcal{TC}^0$ circuit given $\vec{u}$. ∎

We need the following standard $\mathcal{TC}^0$ decoder for Reed-Muller codes.

Lemma 5.4 (Low Depth Decoder for Reed-Muller Code, [AB09, Section 19.3, 19.4])**.** *Let p be a prime, $\mathbb{F} = \mathbb{F}_p$ and $d, m \in \mathbb{N}_{\geq 1}$ such that $d < p/3$. Suppose there is a (hidden) degree-d m-variate polynomial P over $\mathbb{F}$, and let $\delta \in [0, \frac{1}{3(d+1)})$. For any oracle $O \colon \mathbb{F}^m \to \mathbb{F}$ such that*

$$\Pr_{\vec{x} \leftarrow \mathbb{F}^m}[O(\vec{x}) = P(\vec{x})] > 1 - \delta,$$

there is a $\mathcal{P}$-uniform probabilistic $\mathcal{TC}^0$ circuit family $\{\mathsf{RM\text{-}LDC}_{p,m,d}\}_{p,m,d \in \mathbb{N}}$ of size $\mathrm{poly}(m, \log p)$ with non-adaptive O oracle gates, such that for every $\vec{x} \in \mathbb{F}^m$,

$$\Pr[\mathsf{RM\text{-}LDC}^{O}_{p,m,d}(\vec{x}) = P(\vec{x})] \geq 1 - p^{-2m},$$

where the probability is over the randomness of $\mathsf{RM\text{-}LDC}_{p,m,d}$.

Proof. Let $\vec{x} \in \mathbb{F}^m$ be the input; recall that we want to compute $P(\vec{x})$. We will give a randomized non-adaptive oracle $\mathcal{TC}^0$ circuit C^O (with O oracle gates) of size $\mathrm{poly}(m, d, \log p)$ that computes $P(\vec{x})$ with probability at least $2/3$ for every $\vec{x} \in \mathbb{F}^m$. The error probability can then be reduced to p^{-2m}, by running C^O for $O(m \log p)$ times with independent randomness and taking a majority.

We draw a random vector $\vec{\mathbf{v}} \leftarrow \mathbb{F}^m$, and for every $t \in \mathbb{F}$ we define $Q(t) = P(\vec{x} + t \cdot \vec{\mathbf{v}})$. Now for every $t \in [d+1]$ we compute $\alpha_t = O(\vec{x} + t \cdot \vec{\mathbf{v}})$. Letting $\mathbf{z}$ denote the number of $t \in [d+1]$ such that $\alpha_t \neq Q(t)$, we have $\mathbb{E}[\mathbf{z}] \leq \delta(d+1)$ since all α_t distributes uniformly over $\mathbb{F}^m$. By the Markov bound, we have $\Pr[\mathbf{z} = 0] \geq 2/3$.

We then use Lagrange polynomial interpolation to compute a degree-d polynomial $W \colon \mathbb{F} \to \mathbb{F}$ such that $W(t) = \alpha_t$ for all $t \in [d+1]$, and output $W(0)$. Note that since W and Q both have degree at most d, when $\mathbf{z} = 0$, we have $W(0) = Q(0) = P(\vec{x})$, which completes the proof. The whole procedure can be done with $\mathcal{P}$-uniform probabilistic $\mathcal{TC}^0$ oracle circuits of $\mathrm{poly}(m, d, \log p)$-size [HAB02b]. ∎

We are now ready to present the bootstrapping system for highly uniform families of $\mathcal{TC}^0$ circuits. Roughly speaking, the bootstrapping system will be obtained by encoding the polynomials from the polynomial decomposition in Proposition 5.3 by the code from Proposition 4.1.

Proposition 5.5 (refined encoding of efficient polynomial decompositions for highly uniform circuits)**.** *There exists a universal constant $c_1 \in \mathbb{N}$ such that the following holds. Let $\delta \in (0,1)$ be a sufficiently small constant, and let $\{C_n\}_{n \in \mathbb{N}}$ be a δ-highly uniform family of circuits of size $T(n)$ and* *constant depth d. Then, there is a constant κ that only depends on δ such that for every $x \in \{0,1\}^n$ there exists a sequence of functions $w_x^{(1)}, \ldots, w_x^{(d')} \colon [T^\kappa] \to \{0,1\}$, where $T = T(n)$ and $d' = O(d)$, satisfying the following:*

1. **(Faithful representation.)** *There is a $\mathcal{P}$-uniform $\mathcal{TC}^0$ oracle circuit family $\{\mathsf{OUT}_n\}_{n\in\mathbb{N}}$ of size $T^{c_1\cdot\delta}$such that when OUT_n is given $i \in [T]$ as input and oracle access to $w_x^{(d')}$ it outputs $C_n(x)_i$.*
2. **(Base case.)** *There is $\mathcal{P}$-uniform $\mathcal{TC}^0$ circuit family $\{\mathsf{BASE}_n\}_{n\in\mathbb{N}}$ of size $n \cdot T^{c_1\cdot\delta}$ that given $i \in [T^\kappa]$, outputs the description of a $\mathrm{polylog}(n)$-size $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuit C_i such that $\mathsf{C}_i(x)$ outputs $w_x^{(1)}(i)$.*
3. **(Downward self-reducibility.)** *There is a $\mathcal{P}$-uniform $\mathcal{TC}^0$ oracle circuit family $\{\mathsf{DSR}_n\}_{n\in\mathbb{N},\, i\in\{2,\ldots,d'\}}$ of size $T^{c_1\cdot\delta}$ that, when given $j \in [T^\kappa]$ and oracle access to $w_x^{(i-1)}$, outputs $w_x^{(i)}(j)$.*
4. **(Layer reconstruction.)** *There is a $\mathcal{P}$-uniform probabilistic $\mathcal{TC}^0$ oracle circuit family $\{\mathsf{REC}_n\}_{n\in\mathbb{N}}$ that for any $i \in \{2,\ldots,d'\}$ satisfies the following. The circuit REC_n first has a probabilistic preprocessing step, in which it makes non-adaptive queries to $w_x^{(i)}$. Now, fix any $O \colon [T^\kappa] \to \{0,1\}$ such that $\Pr_{j\in[T^\kappa]}[O(j) = w_x^{(i)}(j)] \geq 1/2 + T^{-\delta/c_1}$. Then, with probability at least $1 - 2^{-T^\delta}$ over the coins in the preprocessing step, for any $j \in [T^\kappa]$ it holds that $\Pr\left[\mathsf{REC}_n^O(j) = w_x^{(i)}(j)\right] \geq 1 - 2^{-T^\delta}$, where the probability is over the random coins of REC_n after the preprocessing step.*

Proof. Let $\hat{c}$ be the universal constant from Proposition 5.3. We apply Proposition 5.3 to $\{C_n\}$. Let $p, h, \mathbb{F}$ be as defined in Proposition 5.3. Let κ be a sufficiently large constant that depends on δ. Let c_1 be a sufficiently large constant.

We first define a sequence polynomial $\{P_i\}_{i\in[d']} = \{P_i\}_{i\in[d']}$. We set $d' = m \cdot d + 1$ and

$$\{P_i\}_{i\in[d']} = \{\hat{\alpha}_0, \hat{\alpha}_{1,1}, \ldots, \hat{\alpha}_{1,m}, \hat{\alpha}_{2,1}, \ldots, \hat{\alpha}_{2,2m}, \ldots, \hat{\alpha}_{d,1}, \ldots, \hat{\alpha}_{d,m}\}.$$

By adding dummy variables, we can view all of the polynomials above as mappings from $\mathbb{F}^{2m}$ to $\mathbb{F}$. Note that they all have degree at most $T^{\hat{c}\cdot\delta}$.

Let $N = |\mathbb{F}^{2m}| = p^{2m}$. By the choice of h and m, we have $N = T^{\mu/\delta}$ for a universal constant μ.

Now, we let $w_x^{(1)}$ compute the following Boolean function: given $\vec{u} \in \mathbb{F}^m$ and $i \in \lceil \log p \rceil$, output the i-th bit of the binary representation of $\hat{\alpha}_0(\vec{u})$. (We fill the unused space in $[T^\kappa]$ with zeroes.) The base case follows immediately from the base case of Proposition 5.3.

We instantiate the code Enc from Proposition 4.1 with $\gamma = \frac{2\cdot\hat{c}\cdot\delta^2}{\mu}$ and $\nu = \delta^2$, and let c_0 be the universal constant from Proposition 4.1 and $c^\star = c^\star_{\gamma,\nu}$ be the corresponding constant. We now set κ so that $T^\kappa = N^{c^\star} = \bar{N}$. For every $i \in \{2,\ldots,d'\}$, we define $w_x^{(i)}$ as $\mathsf{Enc}(P_i)$, where we view P_i as a vector from $\mathbb{F}^N$.

Downward self-reducibility. Fix $i \in \{2,\ldots,d'\}$. $\mathsf{DSR}_{n,i}$ operates as follows:

1. Given $j \in [\bar{N}]$ as input, run $Q_N(j)$ to obtain $M = N^\gamma$ many queries $q_1, \ldots, q_M \in [N]$ to P_i, such that $\mathsf{Enc}(P_i)_j = E_N((P_i)_{q_1}, \ldots, (P_i)_{q_M})$.

2. For each $\ell \in [M]$, run the corresponding DSR algorithm that computes P_i with input q_ℓ (interpreted as a vector in $\mathbb{F}^{2m}$) of Proposition 5.3 given an oracle for P_{i-1}; we answer its query to P_{i-1} either using our oracle to $w_x^{(i-1)}$ directly (when $i = 2$), or using $I_N^{w_x^{(i-1)}}$ from the systematic property of Proposition 4.1 (by which $I_N^{\mathsf{Enc}(P_{i-1})}$ computes P_{i-1}).

Since (1) $|Q_N|, |E_N|, |I_N| \le N^{c_0 \cdot (\gamma+\nu)} \le T^{O(\delta)}$ and (2) the DSR $\mathcal{TC}^0$ oracle circuits in Proposition 5.3 and the $\mathcal{TC}^0$ circuit I_N from Proposition 4.1 are both non-adaptive, $\mathsf{DSR}_{n,i}$ can be implemented by a $\mathcal{P}$-uniform non-adaptive $\mathcal{TC}^0$ oracle circuit of size $T^{O(\delta)}$.

Layer reconstruction. Fix $i \in \{2, \ldots, d'\}$ and given an oracle $O\colon \{0,1\}^{\bar{N}} \to \{0,1\}$ such that $\Pr_{j \in [\bar{N}]}[O(j) = \mathsf{Enc}(P_i)] \ge 1/2 + N^{-\nu}$. REC_n operates as follows:

1. Run D_N from Proposition 4.1 with oracle O. We know that with $1 - o(1)$ probability over the preprocessing step of D_N, there is a set $S \subseteq [N]$ with $|S|/N \ge 1 - N^{-\gamma}$ such that $\Pr\left[(D_N)^O(z) = P_i(z)\right] \ge 2/3$ for every $z \in S$ (here we can interpret z as an element in $\mathbb{F}^{2m}$). By running the main step (after the preprocessing step) of D_N for $O(\log N)$ times, each with independent randomness, we can indeed obtain a probabilistic non-adaptive oracle circuit $\bar{D}$ such that for all $z \in S$, $\Pr\left[(\bar{D})^O(z) = P_i(z)\right] \ge 1 - 1/N^2$.

   Hence, by a simple union bound, with probability $1 - o(1)$ over all the randomness above (including the randomness of the main step), we know that $\left(\widetilde{D}_N\right)^O(z) = P_i(z)$ for all $z \in S$, where we use $\widetilde{D}$ to denote $\bar{D}$ with randomness fixed.

2. Now, note that $N^\gamma = \left(T^{\mu/\delta}\right)^{\frac{2\cdot\hat{c}\cdot\delta^2}{\mu}} = T^{2\cdot\hat{c}\cdot\delta}$. In particular, let $d = T^{\hat{c}\cdot\delta}$ be the degree of P_i, we have $N^\gamma = d^2$. Therefore we output $\mathsf{RM\text{-}LDC}_{p,2m,d}^{\widetilde{D}_N^O}(j)$ for the given input $j \in [\bar{N}]$.

By repeating the above procedure $O(T^\delta)$ times and taking the majority answer, we can reduce the error probability to 2^{-T^δ} as desired. Moreover, one can see that REC_N can be implemented by $T^{O(\delta)}$-size probabilistic $\mathcal{TC}^0$ circuits as desired. ■

5.2 Reconstructive targeted HSG for highly uniform $\mathcal{TC}^0$ circuits

Now we are ready to prove Theorem 5.1.

Reminder of Theorem 5.1. *Let $c \in \mathbb{N}_{\ge 1}$ be a sufficiently large universal constant. For every $\gamma \in (0,1)$ and $d \in \mathbb{N}_{\ge 1}$ there exist $d_1 \in \mathbb{N}_{\ge 1}$ and $\delta \in (0,1)$ such that the following holds. Let $T, M\colon \mathbb{N} \to \mathbb{N}$ be such that $M \le T^{\gamma/c}$. Let $f\colon \{0,1\}^n \to \{0,1\}^*$ be computable by a family of δ-highly uniform threshold circuits of depth d and T size. Then, there exist deterministic algorithms $H_f^{\mathtt{CT\text{-}TC}^0}$ and $R_f^{\mathtt{CT\text{-}TC}^0}$ that for every $z \in \{0,1\}^n$ the following hold:*

1. **Generator:** *When given input z, the machine $H_f^{\mathtt{CT\text{-}TC}^0}$ runs in time $\mathrm{poly}(T)$ and prints a set of strings in $\{0,1\}^M$.*

2. **Reconstruction:** $R_f^{\texttt{CT-TC}^0}(1^n)$ *outputs the description of a probabilistic*

$$\left(\mathcal{TC}^0_{d_1}[n \cdot T^\gamma] \mapsto \mathcal{TC}^0_{d_1} \circ \mathsf{SUM}[T^\gamma]\right)$$

oracle circuit $\mathcal{R}_f$*, such that given* $D\colon \{0,1\}^M \to \{0,1\}$ *that* $1/M$*-avoids* $H_f^{\texttt{CT-TC}^0}(z)$ *as oracle, we have*

$$\Pr_{R_f \leftarrow \mathcal{R}_f}\left[R_f^D(z) \text{ outputs a } \mathcal{TC}^0_{d_1} \text{ oracle circuit } E \text{ such that } \mathsf{tt}(E^D) = f(z)\right] \geq 2/3.$$

Proof. Let c_1 be the universal constant from Proposition 5.5. Let $d_1 \in \mathbb{N}_{\geq 1}$ and $\delta \in (0,1)$ to be specified later. Let $\kappa = \kappa(\delta)$ be the corresponding constant from Proposition 5.5. Let $c \in \mathbb{N}_{\geq 1}$ be a sufficient large universal constant.

Without loss of generality, we can assume $M = T^{\gamma/c}$ since for smaller M we can truncate $H_f^{\texttt{CT-TC}^0}$'s outputs to their first M bits and it is straightforward to verify the reconstruction works with minor modifications.

Applying Proposition 5.5 to the δ-highly uniform threshold circuit $\{C_n\}$ of size $T(n)$ and depth d that computes f, for every $z \in \{0,1\}^n$, there is a sequence of functions $w_z^{(1)}, \ldots, w_z^{(d')}\colon [T^{c_1}] \to \{0,1\}$, where $d' = O(d(n))$, that satisfies the conditions in Proposition 5.5.

5.2.1 The generator $H_f^{\texttt{CT-TC}^0}$

We set $\gamma_1 = \frac{\gamma}{c \cdot \kappa}$. We apply Theorem 3.13 with parameter γ_1 and define

$$H_f^{\texttt{CT-TC}^0}(z) = \bigcup_{i \in [d']} G^{\texttt{NW}}(w_z^{(i)}).$$

Note that $H_f^{\texttt{CT-TC}^0}(z)$ outputs a set of string of length $T^{\kappa \cdot \gamma_1} = T^{\gamma/c} = M$, as desired.

Moreover, from the base case and the downward self-reducibility of Proposition 5.5, given z, one can compute $w_z^{(i)}$ for all $i \in [d']$ in $\mathrm{poly}(T)$ time. Since $G^{\texttt{NW}}$ also takes $\mathrm{poly}(T)$ time to compute (Theorem 3.13), we conclude that $H_f^{\texttt{CT-TC}^0}(z)$ can be computed in $\mathrm{poly}(T)$ time as desired.

5.2.2 The reconstruction $R_f^{\texttt{CT-TC}^0}$

We need to output a $\mathcal{TC}^0_{d_1}[n \cdot T^\gamma]$ sampler S that maps randomness to a $\mathcal{TC}^0_{d_1} \circ \mathsf{SUM}[T^\gamma]$ oracle circuit, so that the corresponding probabilistic oracle circuit $\mathcal{R}_f$ satisfies the conditions in the reconstruction part of the theorem.

Notation. Fix an oracle $D\colon \{0,1\}^M \to \{0,1\}$ that $1/M$-avoids $H_f^{\texttt{CT-TC}^0}(z) = \bigcup_{i \in [d']} G^{\texttt{NW}}(w_z^{(i)})$. In particular, it holds that D also $1/M$-distinguishes $G^{\texttt{NW}}(w_z^{(i)})$ for every $i \in [d']$. Let $c_{\texttt{NW}}$ and $d_{\texttt{NW}}$ be the universal constants from Theorem 3.13. Let $S_{\texttt{NW}} = R^{\texttt{NW}}(1^{T^{c_1}})$. Without loss of generality, we assume that $S_{\texttt{NW}}$ takes exactly $r_{\texttt{NW}} = M^{c_{\texttt{NW}}}$ bits as input. Let $d_0, \mu \in \mathbb{N}_{\geq 1}$ be sufficiently large universal constants such that $d_0 \geq d_{\texttt{NW}}$.

High-level overview of the construction. Roughly speaking, we will first construct d' samplers $S_2, \ldots, S_{d'+1}$, such that each S_i maps its own input (*i.e.*, the randomness) to a (deterministic) oracle circuit E_i. The overall sampler S then runs all the S_i with independent randomness, and composes their outputted circuits together to form a single circuit

$$E = E_{d'+1} \circ \cdots \circ E_2.$$

In more detail, for every $i \in \{2, \ldots, d'\}$, E_i takes the output of E_{i-1} ($i > 2$) or z ($i = 2$) as input, and outputs the description of an oracle circuit C_i such that C_i^D is supposed to compute $w_z^{(i)}$. For $i = d'+1$, $S_{d'+1}$ outputs a circuit $E_{d'+1}$ that takes the output of $E_{d'}$ as input and outputs the description of an oracle circuit $C_{d'+1}$ such that $C_{d'+1}^D$ is supposed to computes $f(z)$.

Notation for REC_n. Let $r_{\mathsf{pre}}, r_{\mathsf{main}} \le T^{c_1 \cdot \delta}$ be the number of random bits used by REC_n of Proposition 5.5 for the preprocessing step and the main step, respectively. (We use the main step to denote the operation of REC_n after the preprocessing step.)

Let S_{pre} and S_{main} be the $\mathcal{TC}^0_{d_0}[T^{c_1 \cdot \delta}]$ samplers for the preprocessing step and the main step of REC_n, respectively. Let $i \in [d']$ (note that REC_n does not depend on i). In more detail: (1) S_{pre} takes $\alpha_{\mathsf{pre}} \in \{0,1\}^{r_{\mathsf{pre}}}$ bits as input, and outputs a list of queries to $w_z^{(i)}$, denoted by $q_1, q_2, \ldots, q_t \in [T^{c_1}]$, where $t \le T^{c_1 \cdot \delta}$; (2) S_{main} takes $\alpha_{\mathsf{main}} \in \{0,1\}^{r_{\mathsf{main}}}$ as input, and outputs a $\mathcal{TC}^0_{d_0}$ oracle circuit C'_i of size $T^{c_1 \cdot \delta}$ that takes t bits and $j \in [T^{c_1}]$ as input.

The promise of Proposition 5.5 implies that for any $O\colon \{0,1\}^{T^{c_1}} \to \{0,1\}$ satisfying

$$\Pr_{j \in [T^{c_1}]}[O(j) = w_z^{(i)}(j)] \ge 1/2 + T^{-\delta/c_1},$$

with probability at least $1 - 2^{-T^{\delta}+1} \cdot T^{c_1} \ge 1 - 2^{-T^{\delta}/2}$ over $\alpha_{\mathsf{pre}} \leftarrow U_{r_{\mathsf{pre}}}$ and $\alpha_{\mathsf{main}} \leftarrow U_{r_{\mathsf{main}}}$, it holds that $(C''_i)^O(j) := (C'_i)^O(w_z^{(i)}(q_1), \ldots, w_z^{(i)}(q_t), j)$ computes $w_z^{(i)}$. We set $c \ge \frac{3\gamma \cdot c_1}{\delta}$ so that we have $1/2 + M^{-3} \ge T^{-\delta/c_1}$. (To see this, note that $M^3 = T^{3\gamma/c} \ge T^{\delta/c_1}$ by our choice of c.)

5.2.3 Construction of S_2

We first construct the sampler S_2, whose properties are summarized by the claim below. We remark that the sampled circuit E_2 below does not need an oracle.

Claim 5.6. *There is a polynomial-time algorithm that, given 1^n, outputs a $\mathcal{TC}^0_{O(d_0)}[n \cdot T^{\gamma/2}]$ circuit S_2 satisfying the following:*

1. *S_2 takes $r_2 = n \cdot T^{\gamma/2}$ bits as input, and outputs the description of a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{SUM}$ circuit E_2.*

2. *E_2 takes $z \in \{0,1\}^n$ as input, and outputs the description of a $T^{\mu \cdot c_1 \cdot \delta}$-size $\mathcal{TC}^0_{\mu \cdot d_0}$ oracle circuit C_2.*

3. *For every $z \in \{0,1\}^n$, with probability at least $1 - 1/3d'$ over $E_2 \leftarrow S_2(U_{r_2})$, letting $C_2 = E_2(z)$, it holds that C_2^D computes $w_z^{(2)}$.*

Before proving Claim 5.6. We need the following observation, which follows directly by combining the base case and the properties of DSR_n of Proposition 5.5.

Observation 5.7. *There is a $\mathcal{P}$-uniform $n \cdot T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{d_0}$ circuit that takes input $i \in [T^{c_1}]$ and outputs a $T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{d_0} \circ \mathsf{SUM}$ circuit W_i such that $W_i(z) = w_z^{(2)}(i)$ for all $z \in \{0,1\}^n$.*

Proof of Claim 5.6. S_2 consists of two stages, $S_{2,1}$ and $S_{2,2}$, such that $S_{2,1}$ aims to sample a circuit $E_{2,1}$ that runs the Nisan-Wigderson reconstruction of Theorem 3.13 to obtain an oracle circuit $\widetilde{C}_2$ that weakly approximates $w_z^{(2)}$, and $S_{2,2}$ aims to sample a circuit $E_{2,2}$ that corrects $\widetilde{C}_2$ into another oracle circuit C_2 that computes $w_z^{(2)}$ on all inputs. From now on, we describe $S_{2,1}$ and $S_{2,2}$ separately, and show how S combines them together.

Construction of $S_{2,1}$. $S_{2,1}$ takes $r_{\mathtt{NW}}$ bits as input, denoted by $r_{2,1} \in \{0,1\}^{r_{\mathtt{NW}}}$. $S_{2,1}$ first uses $r_{2,1}$ to compute a circuit $E_{2,1}$ that maps $z \in \{0,1\}^n$ to the description of a $M^{c_{\mathtt{NW}}}$-size $\mathcal{TC}^0_{d_{\mathtt{NW}}}$ oracle circuit $\widetilde{C}_2 = S_{\mathtt{NW}}^{w_z^{(2)}}(r_{2,1})$.

Formally, given $r_{2,1}$, $S_{2,1}$ computes all the queries of $S_{\mathtt{NW}}$ made to $w_z^{(2)}$ in $\mathcal{TC}^0_{d_{\mathtt{NW}}}[M^{c_{\mathtt{NW}}}]$ (note that $S_{\mathtt{NW}}$ is a non-adaptive oracle circuit), and applies Observation 5.7 to replace all calls to $w_z^{(z)}$ in $S_{\mathtt{NW}}$ by $T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{d_0} \circ \mathsf{SUM}$ circuits with input z. This way, $S_{2,1}$ outputs the desired $T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{SUM}$ circuit $E_{2,1}$.

Moreover, by Observation 5.7, we know that $S_{2,1}$ can be implemented by a $n \cdot T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit.

Construction of $S_{2,2}$. Let $r_{2,2} = r_{\mathsf{pre}} + r_{\mathsf{main}}$. $S_{2,2}$ takes $(\alpha_{\mathsf{pre}}, \alpha_{\mathsf{main}}) \in \{0,1\}^{r_{2,2}}$ as input, it first runs $S_{\mathsf{pre}}(\alpha_{\mathsf{pre}})$ to compute $q_1, q_2, \dots, q_t \in [T^{c_1}]$, and then runs $S_{\mathsf{main}}(\alpha_{\mathsf{main}})$ to obtain the oracle circuit C_2', then it constructs the desired circuit $E_{2,2}$ that first computes $w_z^{(2)}(q_1), \dots, w_z^{(2)}(q_t)$, and then outputs C_2'' by fixing the first t bits of the input to C_2' to $w_z^{(2)}(q_1), \dots, w_z^{(2)}(q_t)$. Note that C_2'' is a $T^{c_1\cdot\delta}$-size $\mathcal{TC}^0_{d_0}$ circuit.

By Observation 5.7, $E_{2,2}$ is a $T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{SUM}$ circuit, and $S_{2,2}$ can be implemented by a $n \cdot T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit.

Construction of S_2. Finally, S_2 runs $S_{2,1}$ and $S_{2,2}$ with independent randomness to obtain circuits $E_{2,1}$ and $E_{2,2}$. It then constructs the final circuit E_2 that works as follows: E_2 first runs $E_{2,1}$ and $E_{2,2}$ in parallel (on input z) to obtain the description of the oracle circuit $\widetilde{C}_2$ and the oracle circuit C_2'', and then replaces the oracle in C_2'' by $\widetilde{C}_2$ to obtain the final oracle circuit C_2. Recall that $d_0 \geq d_{\mathtt{NW}}$, C_2 is a $T^{\mu\cdot c_1\cdot\delta}$-size $\mathcal{TC}^0_{\mu\cdot d_0}$ circuit.

With a standard encoding of $\mathcal{TC}^0$ oracle circuits, this oracle replacement operation can be done by a polynomial-size $\mathcal{TC}^0_{O(d_0)}$ circuit (polynomial in terms of the total input length $|\widetilde{C}_2| + |C_2''|$). Hence E_2 is a $T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{SUM}$ circuit, and S_2 can be implemented by a $n \cdot T^{O(c_1\cdot\delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit.

Analysis of S_2. We set δ sufficiently small compared to γ, so that the $T^{O(c_1\cdot\delta)}$ above is at most $T^{\gamma/2}$. The first two items of the claim are established by the discussions above. Now we show the last item. By Theorem 3.13, we know that with probability $1 - 2^{-3M}$ over $E_{2,1} \leftarrow S_{2,1}(\mathcal{U}_{r_{\mathtt{NW}}})$, for $\widetilde{C}_2 = E_{2,1}(z)$, it holds that $\widetilde{C}_2^D$ $(1/2 + M^{-3})$-approximates $w_z^{(2)}$. Then recall that by our choice of c we have $1/2 + M^{-3} \geq 1/2 + T^{-\delta/c_1}$, we have that with probability $1 - 2^{-T^{\delta}/2}$ over $E_{2,2} \leftarrow S_{2,2}(\mathcal{U}_{r_{2,2}})$, for $C_2'' = E_{2,2}(z)$, it holds that $(C_2'')^{\widetilde{C}^D}$ computes $w_z^{(2)}$. A simple union bound completes the proof. ■

5.2.4 Construction of S_i for $i > 2$

Now we construct the sampler S_i for $i \in \{3, \ldots, d'\}$, whose properties are summarized by the claim below. Unlike Claim 5.6, the sampled circuit E_i below needs D as the oracle.

Claim 5.8. *There is a polynomial-time algorithm that, given 1^n and $i \in \{3, \ldots, d'\}$, outputs a $\mathcal{TC}^0_{O(d_0)}\left[T^{\gamma/2}\right]$ circuit S_i satisfying the following:*

1. *S_i takes $r_i = T^{\gamma/2}$ bits as input, and outputs the description of a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit E_i.*
2. *E_i takes the description of a $T^{\mu \cdot c_1 \cdot \delta}$-size $\mathcal{TC}^0_{\mu \cdot d_0}$ oracle circuit C_{i-1} as input, and outputs the description of a $T^{\mu \cdot c_1 \cdot \delta}$-size $\mathcal{TC}^0_{\mu \cdot d_0}$ oracle circuit C_i.*
3. *For every oracle circuit C_{i-1} such that C^D_{i-1} computes $w_z^{(i-1)}$, with probability at least $1 - 1/3d'$ over $E_i \leftarrow S_i(U_{r_i})$, it holds that $C_i = E_i(C_{i-1})$ computes $w_z^{(i)}$ given the oracle D.*

Proof. First, in polynomial-time one can compute a $\mathcal{TC}^0_{O(d_0)}[T^{O(c_1 \cdot \delta)}]$ circuit $E_{i,0}$ that takes the description of a $\mathcal{TC}^0_{\mu \cdot d_0}\left[T^{\mu \cdot c_1 \cdot \delta}\right]$ oracle circuit C_{i-1} such that C^D_{i-1} computes $w_z^{(i-1)}$, composes it with the DSR_n algorithm of Proposition 5.5, and outputs the description of an $\mathcal{TC}^0_{O(d_0)}[T^{O(c_1 \cdot \delta)}]$ oracle circuit F_i such that F_i^D computes $w_z^{(i)}$. Since $E_{i,0}$ does not depend on z, we can hardwire $E_{i,0}$ in to the description of S_i so that S_i can output it directly.

After $E_{i,0}$, similar to S_2, S_i consists of two stages, $S_{i,1}$ and $S_{i,2}$, such that $S_{i,1}$ aims to sample a circuit $E_{i,1}$ that runs the Nisan-Wigderson reconstruction of Theorem 3.13 to obtain an oracle circuit $\widetilde{C}_i$ that weakly approximates $w_z^{(i)}$, and $S_{i,2}$ aims to sample a circuit $E_{i,2}$ that corrects $\widetilde{C}_i$ into another oracle circuit C_i that computes $w_z^{(i)}$ on all inputs. From now on, we describe $S_{i,1}$ and $S_{i,2}$ separately, and show how S_i combines them together.

Construction of $S_{i,1}$. $S_{i,1}$ takes $r_{\mathtt{NW}}$ bits as input, denoted by $r_{i,1} \in \{0,1\}^{r_{\mathtt{NW}}}$. $S_{i,1}$ first uses $r_{i,1}$ to compute a circuit $E_{i,1}$ that maps the description of F_i to the description of a $M^{c_{\mathtt{NW}}}$-size $\mathcal{TC}^0_{d_{\mathtt{NW}}}$ oracle circuit $\widetilde{C}_i = S_{\mathtt{NW}}^{w_z^{(i)}}(r_{i,1})$.

Formally, given $r_{i,1}$, $S_{i,1}$ computes all the queries of $S_{\mathtt{NW}}$ made to $w_z^{(i)}$ in $\mathcal{TC}^0_{d_{\mathtt{NW}}}[M^{c_{\mathtt{NW}}}]$ (note that $S_{\mathtt{NW}}$ is a non-adaptive oracle circuit) and outputs the oracle circuit $E_{i,1}$ that works as follows: $E_{i,1}$ takes the description of the oracle circuit F_i as input, replaces all calls to $w_z^{(z)}$ in $S_{\mathtt{NW}}$ by evaluating F_i^D, and then outputs the description of the resulting circuit $\widetilde{C}_i$.

Note that $S_{i,1}$ can be implemented by a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit, and $E_{i,1}$ is a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ oracle circuit.

Construction of $S_{i,2}$. Let $r_{2,2} = r_{\mathsf{pre}} + r_{\mathsf{main}}$. $S_{i,2}$ takes $(\alpha_{\mathsf{pre}}, \alpha_{\mathsf{main}}) \in \{0,1\}^{r_{2,2}}$ as input. It first runs $S_{\mathsf{pre}}(\alpha_{\mathsf{pre}})$ to computes $q_1, q_2, \dots, q_t \in [T^{c_1}]$, and then runs $S_{\mathsf{main}}(\alpha_{\mathsf{main}})$ to obtain the oracle circuit C'_2, then it constructs the desired circuit $E_{i,2}$ that takes the description of F_i as input, computes $w_z^{(i)}(q_1), \dots, w_z^{(i)}(q_t)$ by evaluating F_i, and then outputs C''_i by fixing the first t bits of the input to C'_i to $w_z^{(i)}(q_1), \dots, w_z^{(i)}(q_t)$. Note that $E_{i,2}$ is a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit, and $S_{i,2}$ can be implemented by a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit.

Construction and analysis of S_i. Finally, S_i runs $S_{i,1}$ and $S_{i,2}$ with independent randomness to obtain circuits $E_{i,1}$ and $E_{i,2}$. It then constructs the final circuit E_i that works as follows: E_i first runs $E_{i,0}$ with input C_{i-1} to obtain the oracle circuit F_i, then it runs $E_{i,1}$ and $E_{i,2}$ with input F_i in parallel to obtain the description of the oracle circuit $\widetilde{C}_i$ and the oracle circuit C''_i, and then replace the oracle in C''_i by $\widetilde{C}_i$ to obtain the final oracle circuit C_i.

Similarly to the proof of Claim 5.6, this oracle replacement operation can be done by a polynomial-size $\mathcal{TC}^0_{d_0}$ circuits. Hence E_i is a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit, and S_i can be implemented by a $T^{O(c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(d_0)}$ circuit.

The analysis follows from the same argument of Claim 5.6. ■

5.2.5 Final construction

Finally, using the OUT_n circuit from Proposition 5.5, in polynomial time we can compute a $\mathcal{TC}^0_{O(d_0)}[T^{O(c_1 \cdot \delta)}]$ circuit $E_{d'+1}$ that takes the description of a $T^{\mu \cdot c_1 \cdot \delta}$-size $\mathcal{TC}^0_{\mu \cdot d_0}$ oracle circuit $C_{d'}$ as input, and outputs the description of a $T^{O(\mu \cdot c_1 \cdot \delta)}$-size $\mathcal{TC}^0_{O(\mu \cdot d_0)}$ oracle circuit $C_{d'+1}$, such that if $C^D_{d'}$ computes $w_z^{(d')}$, then $C^D_{d'+1}$ computes $f(z)$. Since the above algorithm is deterministic, we can construct a $\mathcal{TC}^0_{O(d_0)}[T^{O(c_1 \cdot \delta)}]$ circuit $S_{d'+1}$ that takes no input and outputs $E_{d'+1}$ as the "sampler" for the last stage. (We define $S_{d'+1}$ only for notational convenience.)

As already discussed in the high-level overview, the final sampler S runs $S_2, \dots, S_{d'+1}$ with independent randomness to obtain circuits $E_2, \dots, E_{d'+1}$. The final output of S is then

$$E = E_{d'+1} \circ E_{d'} \circ \cdots \circ E_2.$$

By setting δ small enough and d_1 large enough, the desired complexity on E and S follows from Claim 5.6 and Claim 5.8, and the correctness follows from a union bound. ■

6 Derandomization vs refutation

In this section we prove our main results, relying on the technical tools that were developed in previous sections. First, in Section 6.1, we prove Theorem 1.1. Then, in Section 6.2, we prove Theorems 1.2, 1.3 and 1.4. Finally, in Section 6.3, we prove Theorem 1.8.

6.1 Derandomization vs refutation against low-space streaming algorithms

Let us start by proving the direction "refutation $\Rightarrow$ derandomization". That is, we show that deterministically refuting low-space streaming algorithms implies that $pr\mathcal{BPP} = pr\mathcal{P}$.

Theorem 6.1 (refutation of streaming algorithms implies derandomization)**.** *Let $\varepsilon \in (0,1)$, let $T(N) \geq N$ and $p(n)$ be polynomials, and let f be a p-bounded T-time algorithm-dependent hard function for* str-$\mathcal{TISP}[T^{1+\varepsilon}, n^{\varepsilon}]$. *Assume that there exists a $\mathcal{P}$-computable N^{ε}-compression list-refuter for f against* str-$\mathcal{TISP}[T^{1+\varepsilon}, n^{\varepsilon}]$*. Then, $pr\mathcal{BPP} = pr\mathcal{P}$.*

Proof. To prove the theorem, it suffices to show that for every linear-time machine M, given input $x \in \{0,1\}^m$, we can distinguish between the case that $\Pr_r[M(x,r) = 1] \geq 1/2$ and $\Pr_r[M(x,r) = 1] = 0$. Without loss of generality, we assume that M uses exactly m bits of randomness.

Notation. We begin by introducing some notation. Let M be a probabilistic linear-time machine, and let c be the universal constant from Theorem 3.15. Let $\delta = \varepsilon/4c$. We set $n = m^{1/\delta}$. For every $a \in \{0,1\}^m$, we use $\bar{a} \in \{0,1\}^n$ to denote the padded string $\bar{a} = (a, 0^{n-m})$. We also set $\gamma = \gamma(\delta)$ be such that $T(N)^{\gamma} = N^{\varepsilon/4}$.

Let $(a,x) \in \{0,1\}^m \times \{0,1\}^{p(n)}$ and $N = |(\bar{a},x)| = n + p(n)$. We also set $S_{a,x} = H_f^{\mathtt{CT}}(\bar{a},x)$, with parameter γ and output length m. (Note that $m = n^{\varepsilon/4c} < T(N)^{\gamma/c} = N^{\varepsilon/4c}$, so the assumption of Theorem 3.15 is satisfied.)

Lemma 6.2 (instance-wise reconstruction)**.** *There is a one-pass streaming algorithm $R = R_f$ (i.e., the algorithm depends on f) that uses space N^{ε} and time $T(N)^{1+\varepsilon}$ and satisfies the following. For any fixed $(a,x) \in \{0,1\}^m \times \{0,1\}^{p(n)}$, if*

$$\Pr_r[M(a,r) = 1] \geq 1/2 \text{ and } \Pr_{s \in S_{a,x}}[M(a,s) = 1] = 0\,,$$

then, when R is given input $(\bar{a},x)$, with probability at least $2/3$ it prints a circuit of size N^{ε} whose truth-table is $f(\bar{a},x)$.

Proof. Let $z = (\bar{a},x)$. We define R to be the reconstruction algorithm $R_f^{\mathtt{CT}}$ from Theorem 3.15 with oracle replaced by $D_a(r) := M(a,r)$. From the assumption we know that $D_a(\cdot)$ 1/2-avoids $H_f^{\mathtt{CT}}(z) = S_{a,x}$, we know that with probability at least $2/3$, $R_f^{\mathtt{CT}}$ outputs an oracle circuit $C_{f(z)}$ of size $T(N)^{\gamma} = N^{\varepsilon/4}$ such that the truth-table of $C_{f(z)}^{D_a}$ is $f(z)$. R then simply composes $C_{f(z)}$ with D_a to output a circuit of $N^{\varepsilon/4} \cdot m^2 \leq N^{\varepsilon}$. This establishes the correctness of R.

Now we verify the time and space complexity of R. From Theorem 3.15, R is a one-pass streaming algorithm that runs in $m^{c+1} \cdot T(N)^{1+\gamma} \leq T(N)^{1+\varepsilon}$ time and uses at most $m^c \leq N^{\varepsilon}$ space. This completes the proof. ■

Now we are ready to prove the theorem. Given input $a \in \{0,1\}^m$ for M, we run the list-refuter on input $(R, \bar{a})$ to obtain $x_1, ..., x_k \in \{0,1\}^{p(n)}$, where $k = \mathrm{poly}(n)$. For each $i \in [k]$, we compute the list $S_i = S_{a,x_i}$, and finally we output $\bigvee_{i \in [k], s \in S_i} M(a,s)$. From Theorem 3.15, the whole procedure can be done in $\mathrm{poly}(n)$ time.

Assume towards a contradiction that for some $a \in \{0,1\}^m$ it holds that

$$\Pr_{r \in \{0,1\}^m}[M(a,r) = 1] \geq 1/2 \text{ and } \bigvee_{i \in [k], s \in S_i} M(a,s) = 0\,.$$

By Lemma 6.2, for every $i \in [k]$ it holds that $R(\bar{a}, x_i)$ prints, with high probability, a circuit of size N^{ε} whose truth-table is $f(\bar{a}, x_i)$. This contradicts the properties of the compression list-refuter. ■

We now prove the converse direction, which asserts that derandomization implies refutation. Recall that the deduced refuter in Theorem 1.1 works not only for streaming algorithms, but for essentially any class of RAMs, where the class only needs to satisfy a very weak property. Let us define this property and prove the result.

Definition 6.3 (closure under error-reduction)**.** *We say that a class $\mathcal{C}$ of probabilistic RAMs is* closed under error-reduction *if there is a deterministic polynomial-time algorithm that takes as input a description of any $M \in \mathcal{C}$ and outputs a description of M' such that $M'(x)$ runs $M(x)$ for 100 times with independent coins each time, and outputs the most frequent outcome (breaking ties arbitrarily).*

Theorem 6.4 (derandomization implies refutation)**.** *Let $\mathcal{C}$ be a class of probabilistic RAMs closed under error-reduction, let p be a polynomial, and let $f \in \mathcal{FP}$ be a p-bounded algorithm-dependent hard function for $\mathcal{C}$ that admits a $\mathcal{BPP}$-refuter. Assuming $pr\mathcal{P} = pr\mathcal{BPP}$, there is an $\mathcal{FP}$-refuter for $\mathcal{C}$ against f.*

Proof. Let Ref be the $\mathcal{BPP}$-refuter for f against $\mathcal{C}$. Given input (M, a) where $M \in \mathcal{C}$, let $M' \in \mathcal{C}$ be the error-reduced version of M from Definition 6.3. We construct the circuit

$$D(r, r') = \mathbf{1}\left[M'((a, x), r') \neq f(a, x)\right], \text{ where } x = \mathsf{Ref}((M', a), r) ;$$

that is, D takes as input random coins r for Ref and random coins r' for M'; it runs Ref on input (M', a) with random coins r, to obtain an input x for M'; then it runs M' on input (a, x) with random coins r'; and finally, it compares the output of M' on x to $f(a, x)$.

Since Ref is a $\mathcal{BPP}$-refuter, with probability at least $2/3$ over r, the output x satisfies $\Pr_{r'}[M((a, x), r') = f(a, x)] < 2/3$. Thus, $\Pr_{r,r'}[D(r, r') = 1] \geq (2/9)$. Running the search-to-decision reduction from Theorem 3.16 on the circuit D,[39] we find r^* such that $\Pr_{r'}[D(r^*, r') = 1] \geq 1/9$. Equivalently, denoting $x^* = \mathsf{Ref}((M', a), r^*)$, we have that

$$\Pr_{r'}\left[M'((a, x^*), r') \neq f(a, x^*)\right] \geq 1/9 .$$

The output of the deterministic refuter is x^*.

Now, assume towards a contradiction that $\Pr_{r''}[M((a, x^*), r'') = f(a, x^*)] \geq 2/3$. Then, by the definition of M' as the error-reduced version of M, we have that $\Pr_{r'}[M'((a, x^*), r') = f(a, x^*) \geq 0.99$. This yields a contradiction. ■

The following corollary is a more general version of Theorem 1.1, and it asserts an equivalence between refutation and derandomization.

Corollary 6.5. *The following statements are equivalent:*

1. *For some $\varepsilon > 0$ and polynomials p, T and a p-bounded T-time algorithm-dependent hard function f against $\mathsf{str}\mathcal{TISP}[T(n)^{1+\varepsilon}, n^{\varepsilon}]$, there there is an N^{ε}-compression list-refuter in $\mathcal{FP}$ for f against $\mathsf{str}\mathcal{TISP}[T(n)^{1+\varepsilon}, n^{\varepsilon}]$.*

2. *$pr\mathcal{BPP} = pr\mathcal{P}$.*

[39] By our assumption that $pr\mathcal{BPP} = pr\mathcal{P}$, it follows that CAPP for general circuits is solvable in deterministic polynomial time, and hence an algorithm as in the hypothesis of Theorem 3.16 exists.

3. *For every class $\mathcal{C}$ of probabilistic RAMs closed under error-reduction, and any p-bounded algorithm-dependent hard function $f \in \mathcal{FP}$ for $\mathcal{C}$ that admits a $\mathcal{BPP}$-refuter (where p is a polynomial), there is an $\mathcal{FP}$-refuter for f against $\mathcal{C}$.*

Proof. The implication (1) $\Rightarrow$ (2) follows from Theorem 6.1. The implication (2) $\Rightarrow$ (3) follows from Theorem 6.4. For the implication (3) $\Rightarrow$ (1), it suffices to show, unconditionally, that there is a function f computable in polynomial time T that is hard against str-$\mathcal{TISP}[T^{1+\varepsilon}, n^{\varepsilon}]$, and that *has a $\mathcal{BPP}$-refuter*. (Note that we will be using a standard hard function, which is a special case of an algorithm-dependent hard function.)

Such a function indeed exists, because the well-known lower bounds for functions in $\mathcal{FP}$ against streaming algorithms of sublinear space complexity (and any time complexity) actually hold *on average*. That is, the classical proofs define very simple distributions, and show that with probability $\Omega(1)$ over an input sampled from these distributions, the streaming algorithm fails on that input.[40] Thus, the $\mathcal{BPP}$-refuter can repeatedly sample an input and verify that the streaming algorithm fails to compute the hard function on it (until it finds a suitable input). ∎

Finally, recall that in Section 1.3 we mentioned that proving a statement along the lines of "$\mathcal{BPP}$-refuters imply derandomization" would unconditionally imply that $pr\mathcal{BPP} = pr\mathcal{P}$. Let us now state this claim formally and prove it.

Claim 6.6. *Let $\mathcal{C}$ be any class of RAMs running in polynomial time such that for every $M \in \mathcal{C}$ and every input z there is a string y such that $\Pr[M(z) = y] \geq 2/3$. Consider the following statement:*

> (Cond.Stt.) *Assume that there is a probabilistic polynomial-time RAM f and a deterministic polynomial-time algorithm R such that for every $M \in \mathcal{C}$ and sufficiently large $n \in \mathbb{N}$ and $a \in \{0,1\}^n$, the algorithm $R(M, a)$ prints $x \in \{0,1\}^{\mathrm{poly}(n)}$ satisfying $\Pr[M(x,a) = f(x,a)] < 2/3$, where the probability is over the random coins of M and of f. Then, $pr\mathcal{BPP} = pr\mathcal{P}$.*

Then, we have that

$$(\mathsf{Cond.Stt.}) \implies pr\mathcal{BPP} = pr\mathcal{P} \; .$$

In other words, to prove that $pr\mathcal{BPP} = pr\mathcal{P}$, it suffices to prove the conditional statement (Cond.Stt.).

Proof. For any $\mathcal{C}$, we show that f and R as in the hypothesis of (Cond.Stt.) exist unconditionally. Thus, if (Cond.Stt.) is true, then $pr\mathcal{BPP} = pr\mathcal{P}$.

To see this, let T be the polynomial bound on the running times of machines in $\mathcal{C}$, and consider the following machine f. Given as input (x, a), simulate the first $\ell = \log^*(n)$ RAMs $M_1, ..., M_\ell$ on input (x, a). Specifically, each machine is simulated for T steps, and we repeat the simulation for $O(\log(\ell))$ times (so that if there exists y such that $\Pr[M_i(a, x) = y] \geq 2/3$, then with probability at least $1/(100\ell)$, this y will be the output of M_i in at least 0.6 of its simulations). For each $i \in [\ell]$, denote by $y^{(i)}$ the output that M_i prints in at least 0.6 of its simulations (if no such string exists, or if M_i does not halt after $T^{1+\varepsilon}$ steps in one of the simulations, then $y^{(i)} = 0^\ell$).

[40] For example, the lower bound in [AMS99, Proposition 3.1] holds with probability $\Omega(1)$ over a distribution that is obtained by applying a polynomial-time transformation to the hard distribution from the proof of the communication lower bound for disjointness [Raz92]. Alternatively, one can directly consider the latter lower bound as a lower bound on streaming algorithms (where the streaming algorithm first sees Alice's input x, bit-by-bit, and then sees Bob's input y, bit-by-bit), in which case the hard distribution from [Raz92] is also hard for streaming algorithms.

Let $z_i = \begin{cases} y_i^{(i)} & |y^{(i)}| \geq i \\ 0 & o.w. \end{cases}$. Finally, print the ℓ-bit string such that for every $i \in [\ell]$ it holds that $f(x,a)_i = \neg z_i$.

Note that f runs in probabilistic polynomial time. Also note that for every $M \in \mathcal{C}$ there are at most finitely many inputs (x,a) such that $\Pr[M(x,a) = f(x,a)] \geq 2/3$. (Recall that, by the definition of $\mathcal{C}$, for every $M \in \mathcal{C}$ and every input (x,a) there exists y such that $\Pr[M(x,a) = y] \geq 2/3$.) Hence, there is a trivial algorithm R that satisfies the hypothesis, namely the algorithm that gets input (M,a) and outputs any fixed x (e.g., $x = 0^{p(|a|)}$). By the conditional statement (Cond.Stt.), it follows that $pr\mathcal{BPP} = pr\mathcal{P}$. ■

6.2 Derandomization vs refutation for $\mathcal{TC}^0$

In this section we present connections between refutation and derandomization in the setting of weak circuit classes, and in particular for $\mathcal{TC}^0$. In Section 6.2.1 we present the results concerning refuting $\mathsf{Identity}$ (i.e., Theorem 1.2), and in Section 6.2.2 we present the results concerning refuting any function in highly uniform $\mathcal{TC}^0$ (i.e., Theorems 1.3 and 1.4).

6.2.1 Special case: derandomization vs refutation for $\mathsf{Identity}$ against $\mathcal{TC}^0$

Let us prove Theorem 1.2, which asserts an equivalence between refuting $\mathsf{Identity}$ against small probabilistic $\mathcal{TC}^0 \circ \oplus$ circuits, and derandomization of $\mathcal{TC}^0$. As a first step, we prove that compression-refuters for probabilistic $\mathcal{TC}^0 \circ \oplus$ circuits with n^ε gates suffices for derandomization:

Theorem 6.7 (compression refutation for $\mathsf{Identity}$ against small probabilistic $\mathcal{TC}^0$ circuits implies derandomization)**.** *For every $d \in \mathbb{N}_{\geq 1}$ there exists $d' \in \mathbb{N}_{\geq 1}$ such that the following holds. Assume the following:*

- *For some $\varepsilon \in (0,1)$, there is a $\mathcal{P}$-computable $(\mathcal{TC}^0_{d'}, n^\varepsilon)$-compression refuter for* $\mathsf{Identity}$ *against probabilistic $(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto (\mathcal{TC}^0_{d'} \circ \mathsf{XOR})[n^\varepsilon])$-circuits.*

Then, there is a deterministic polynomial-time algorithm solving CAPP *for $\mathcal{TC}^0_d$ circuits.*

Proof. Fix $d \in \mathbb{N}_{\geq 1}$. Given a $\mathcal{TC}^0_d$ circuit $C\colon \{0,1\}^m \to \{0,1\}$. By adding dummy inputs, we can assume C has size n as well. Our goal is to estimate $\Pr_{z \in \{0,1\}^m}[C(z) = 1]$ within an additive error of $1/m$.

Let $\varepsilon \in (0,1)$. Let c_{STV} and d_{STV} be the universal constants from Theorem 3.14, and let $\gamma = \varepsilon/4c_{\mathsf{STV}}$, and $n = m^{1/\gamma}$. We instantiate Theorem 3.14 with parameter γ. And we run $R^{\mathsf{STV}}(1^n)$ to obtain the description of a probabilistic

$$\left(\mathcal{TC}^0_{d_{\mathsf{STV}}}[n \cdot m^{c_{\mathsf{STV}}}] \mapsto \mathcal{TC}^0_{d_{\mathsf{STV}}} \circ \mathsf{XOR}[m^{c_{\mathsf{STV}}}]\right)$$

oracle circuit $\mathcal{R}'$, such that for every $a \in \{0,1\}^n$, given $D\colon \{0,1\}^m \to \{0,1\}$ that $1/m$-distinguishes $G^{\mathsf{STV}}(a)$ as oracle, we have

$$\Pr_{R' \leftarrow \mathcal{R}'}\left[\left(R'\right)^D(a) \text{ outputs a } \mathcal{TC}^0_{d_{\mathsf{STV}}} \text{ oracle circuit } E \text{ such that } \mathrm{tt}(E^D) = a\right] \geq 2/3.$$

Now, noting that $m^{c_{\mathsf{STV}}} = n^{\varepsilon/4}$, we replace the oracle of $\mathcal{R}'$ by our m-size $\mathcal{TC}^0_d$ circuit C to obtain the description of a probabilistic

$$\left(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto \mathcal{TC}^0_{d'} \circ \mathsf{XOR}[n^{\varepsilon}]\right)$$

circuit $\mathcal{R}$ for some constant d' that only depends on $d_{\mathtt{STV}}$ and d.

We next run the assumed $\mathcal{P}$-computable $(\mathcal{TC}^0_{d'}, n^{\varepsilon})$-compression refuter for $\mathsf{Identity}$ on $\mathcal{R}$ to obtain a bad input $a \in \{0,1\}^n$. From the construction of $\mathcal{R}$, we know that C does not $1/m$-distinguishes $G^{\mathsf{STV}}(a)$. Therefore, we can enumerate all outputs of $G^{\mathsf{STV}}(a)$ to estimate $\Pr_{z \in \{0,1\}^m}[C(z) = 1]$ within an additive error of $1/m$. This completes the proof. ∎

Towards proving Theorem 1.2, we want to show that derandomization of $\mathcal{TC}^0$ follows from refuters against small probabilistic $\mathcal{TC}^0 \circ \oplus$ circuits, rather than from compression-refuters against such circuits. This statement seems obvious, since a refuter is intuitively stronger than a list-refuter: given a circuit C whose truth-table is $f(x)$, we can print $f(x)$ by printing the truth-table of C (thus, if we have x such that $f(x)$ cannot be printed by small circuits, then $f(x)$ also cannot be compressed by small circuits). But the point is that the foregoing transformation has computational overheads, which strengthen the circuit model that needs to be refuted.

Thus, we now prove a corollary asserting that derandomization of $\mathcal{TC}^0$ follows from (standard, non-compression) refuters against small probabilistic $\mathcal{TC}^0 \circ \oplus$ circuits, while accounting for this overhead. Recall that we use $\mathcal{TC}^0_d$-$\mathsf{WIRES}[S] \circ \ell$-$\mathsf{XOR}$ to denote a circuit consists with a top $\mathcal{TC}^0_d$ circuit of S total wires and a bottom layer of ℓ parity gates. Then:

Corollary 6.8 (refutation for $\mathsf{Identity}$ against small probabilistic $\mathcal{TC}^0$ circuits implies derandomization)**.** *For every $d \in \mathbb{N}_{\geq 1}$ there exists $d' \in \mathbb{N}_{\geq 1}$ such that the following holds. Assume the following:*

- *For some $\varepsilon \in (0,1)$, there is a $\mathcal{P}$-computable refuter for* $\mathsf{Identity}$ *against probabilistic $(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto (\mathcal{TC}^0_{d'}$-$\mathsf{WIRES}[n^{1+\varepsilon}] \circ n^{\varepsilon}$-$\mathsf{XOR})$-circuits.*

Then, there is a deterministic polynomial-time algorithm solving CAPP *for $\mathcal{TC}^0_d$ circuits.*

Proof. Let $\varepsilon_1 \in (0,1)$ be a constant to be specified later. We first apply Theorem 6.7 with parameters ε_1 and d, and let d'_1 be the corresponding constants. Let $\mu \in \mathbb{N}$ be a sufficiently large constant.

Given the description of a probabilistic $(\mathcal{TC}^0_{d'_1}[n^{1+\varepsilon_1}] \mapsto (\mathcal{TC}^0_{d'_1} \circ \mathsf{XOR})[n^{\varepsilon_1}])$ circuit $\mathcal{C}$, in polynomial-time we can construct the description of a probabilistic

$$\left(\mathcal{TC}^0_{\mu \cdot d'_1}[n^{1+\mu\cdot\varepsilon_1}] \mapsto (\mathcal{TC}^0_{\mu \cdot d'_1}\text{-}\mathsf{WIRES}[n \cdot n^{\mu\cdot\varepsilon_1}] \circ n^{\varepsilon_1}\text{-}\mathsf{XOR}\right)$$

circuit $\mathcal{C}'$, such that $\mathcal{C}'$ first runs $\mathcal{C}$ and treats its output as the description of a $\mathcal{TC}^0_{d'_1}$ circuit E of n^{ε_1} size, and outputs the first n bits of E's truth-table.[41]

Let $d' = \mu \cdot d'_1$ and $\varepsilon = \mu \cdot \varepsilon_1$. From the above transformation, it follows that a $\mathcal{P}$-computable refuter for $\mathsf{Identity}$ against probabilistic $\left(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto \mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[n^{1+\varepsilon}] \circ n^{\varepsilon}\text{-}\mathsf{XOR}\right)$-circuits implies a $\mathcal{P}$-computable $(\mathcal{TC}^0_{d'_1}, n^{\varepsilon_1})$-compression refuter for $\mathsf{Identity}$ against probabilistic $(\mathcal{TC}^0_{d'_1}[n^{1+\varepsilon_1}] \mapsto (\mathcal{TC}^0_{d'_1} \circ \mathsf{XOR})[n^{\varepsilon_1}])$-circuits. The corollary then follows from Theorem 6.7. ∎

[41] More precisely, $\mathcal{C}'$ is the composition of $\mathcal{C}$ and a $\mathcal{TC}_{O(d'_1)}$ circuit U that takes the description of a $\mathcal{TC}^0_{d'_1}$ circuit E of n^{ε_1} size as input, and outputs the first n bits of E's truth-table. It it easy to see that U has $n \cdot n^{O(\varepsilon_1)}$ wires.

We now complement Corollary 6.8 by proving a converse direction (i.e., "derandomization $\Rightarrow$ refutation"), which will complete the proof of Theorem 1.2.

Theorem 6.9 (Theorem 1.2, formally stated)**.** *The following two statements are equivalent:*

1. *For every $d \in \mathbb{N}$, there is a polynomial-time algorithm solving* CAPP *for $\mathcal{TC}^0_d$ circuits.*

2. *For every $d' \in \mathbb{N}$, there exist $\varepsilon \in (0,1)$ and a $\mathcal{P}$-computable refuter for* Identity *against probabilistic $(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto (\mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[n^{1+\varepsilon}] \circ n^{\varepsilon}\text{-}\mathsf{XOR})$-circuits.*

Proof. The direction $(2) \implies (1)$ follows immediately from Corollary 6.8. So it suffices to show the $(1) \implies (2)$ direction.

Fix $d' \in \mathbb{N}$. For convenience, we use $\mathfrak{F}$ to denote probabilistic $(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto (\mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[n^{1+\varepsilon}] \circ n^{\varepsilon}\text{-}\mathsf{XOR})$-circuits.

We first note that given the description of an n-input $\mathfrak{F}$ circuit $\mathcal{C}$, in polynomial time we can construct a $\mathcal{TC}^0$ circuit B such that for every $x \in \{0,1\}^n$, we have $\mathcal{C}(x)$ has the same distribution as $B(x, \mathbf{U}_{r_1})$, where $r_1 \le \mathrm{poly}(n)$. We note that since $\mathcal{C}$ only has n^{ε} gates at the bottom, we have $\Pr_{\alpha \leftarrow \{0,1\}^n}[\mathcal{C}(\alpha) = \alpha] \le 0.01$. We construct the following $\mathcal{TC}^0$ circuit $W \colon \{0,1\}^n \times \{0,1\}^{r_1} \to \{0,1\}$ as

$$W(\alpha, \beta) = \mathbf{1}\Big[B(\alpha, \beta) = f(\alpha)\Big].$$

We know that $\Pr_{\alpha,\beta}[W(\alpha,\beta) = 1] < 0.01$. From (1) and Theorem 3.16, in polynomial deterministic time we can find an $\alpha \in \{0,1\}^{r_2}$ such that $\Pr_{\beta}[W(\alpha,\beta) = 1] < 2/3$, then α is the output of our deterministic refuter. ■

6.2.2 Generalization to any hard function computable by highly uniform $\mathcal{TC}^0$ circuits

In Section 6.2.1 we proved results focusing on refuters against small probabilistic $\mathcal{TC}^0$ circuits for the "hard function" $f =$ Identity. In this section we broaden the class of hard functions f, from Identity to all functions computable in highly uniform $\mathcal{TC}^0$. To do so we will crucially rely on Theorem 5.1. We start by proving (a more general and technical version of) Theorem 1.3.

Theorem 6.10 (compression refutation against small probabilistic $\mathcal{TC}^0$ circuits implies derandomization)**.** *For every $\varepsilon \in (0,1)$ and $d, d_f, k \in \mathbb{N}_{\ge 1}$ there exist $d' \in \mathbb{N}_{\ge 1}$ and $\delta \in (0,1)$ such that the following holds. Let $f \colon \{0,1\}^* \to \{0,1\}^*$ be any function computable by a family of δ-highly uniform threshold circuits of depth d_f and n^k size. Assume the following:*

- *There is a $\mathcal{P}$-computable $(\mathcal{TC}^0_{d'}, n^{\varepsilon})$-compression list-refuter for f against probabilistic $(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}] \mapsto (\mathcal{TC}^0_{d'} \circ \mathsf{SUM})[n^{\varepsilon}])$-circuits.*

Then, there is a deterministic polynomial-time algorithm solving $\mathsf{CAPP}_{0,1/2}$ for $\mathcal{TC}^0_d$ circuits.

Proof. Let $T(n) = n^k$, and let γ be such that $T(n)^{\gamma} = n^{\varepsilon/4}$. Let c be the universal constant from Theorem 5.1. Let d_1 and δ be the corresponding parameters from Theorem 5.1 when applying it with γ and d_f.

Let $f \colon \{0,1\}^* \to \{0,1\}^*$ be a function computable by a family of δ-highly uniform threshold circuits of depth d_f and $T(n)$ size.

Given an m-input $\mathcal{TC}^0_d$ circuit $C \colon \{0,1\}^m \to \{0,1\}$, our goal is to decide between the case that $\Pr_r[C(r) = 1] \ge 1/2$ and $\Pr_r[C(r) = 1] = 0$. By adding dummy gates, without loss of generality, we can assume C can be described by an m-bit string $a \in \{0,1\}^m$, and we also use $C_a \colon \{0,1\}^m \to \{0,1\}$ to denote the circuit corresponds to a. Let $n = m^{4c/\varepsilon}$ so that $m = n^{\varepsilon/4c}$.

Applying Theorem 5.1 with function f, parameter γ and output length m, for $x \in \{0,1\}^n$, we set $S_x = H_f^{\mathtt{CT}}(x)$. (Note that $m = n^{\varepsilon/4c} = T(n)^{\gamma/c}$, so the assumption of Theorem 5.1 is satisfied.)

Lemma 6.11 (instance-wise reconstruction)**.** *There is a constant $d' \in \mathbb{N}$ that only depends on d, f_1, and d_1 such that $d' \geq \max(d_f, d, d_1)$ and the following holds. Given $a \in \{0,1\}^m$, there is a polynomial-time algorithm that computes the description of a $\left(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}]\right)$-samplable probabilistic $\mathcal{TC}^0_{d'} \circ \mathsf{SUM}$ n-input circuit R_a of size n^ε, such that for every $x \in \{0,1\}^n$, if*

$$\Pr_r[C_a(r) = 1] \geq 1/2 \text{ and } \Pr_{s \in S_x}[C_a(s) = 1] = 0\,,$$

then, when R is given input x, with probability at least $2/3$ it prints a $\mathcal{TC}^0_{d'}$ circuit of size n^ε whose truth-table is $f(x)$.

Proof. Let $R_f = R_f^{\mathtt{CT\text{-}TC^0}}(1^n)$ be the $\left(\mathcal{TC}^0_{d_1}[n \cdot T^\gamma]\right)$-samplable probabilistic $\mathcal{TC}^0_{d_1} \circ \mathsf{SUM}$ oracle circuit R_f of size T^γ outputted by $R_f^{\mathtt{CT\text{-}TC^0}}$ from Theorem 5.1. We replace the oracle of R_a by C_a to obtain R_a. Recalling that $m = n^{\varepsilon/4c}$ and $T^\gamma = n^{\varepsilon/4}$, R_a corresponds to a $\left(\mathcal{TC}^0_{d'}[n^{1+\varepsilon}]\right)$-samplable probabilistic $\mathcal{TC}^0_{d'} \circ \mathsf{SUM}$ n-input circuit of size n^ε, for a sufficiently large d' that only depends on d_1, d_f, and d. And from its construction, R_a can be computed from a in polynomial time.

From Theorem 5.1, if $\Pr_r[C_a(r) = 1] \geq 1/2$ and $\Pr_{s \in S_x}[C_a(s) = 1] = 0$, then it holds that $R_a(x)$ prints a $\mathcal{TC}^0_{d'}$ circuit of size n^ε whose truth-table is $f(x)$ with probability at least $2/3$. □

Now, given input $a \in \{0,1\}^m$ to $\mathsf{CAPP}_{0,1/2}$, we construct R_a from Lemma 6.11, and run the compression list-refuter on input $(1^n, R_a)$ to obtain $x_1, ..., x_t \in \{0,1\}^n$, where $t \leq \mathrm{poly}(n)$. For each $i \in [t]$, we compute the list $S_i = S_{x_i}$, and finally we output $\bigvee_{i \in [t], s \in S_i} C_a(s)$. From Theorem 5.1, the whole procedure runs in polynomial time.

Assume towards a contradiction that for some $a \in \{0,1\}^m$ it holds that

$$\Pr_{r \in \{0,1\}^m}[C_a(r) = 1] \geq 1/2 \text{ and } \bigvee_{i \in [t], s \in S_i} C_a(s) = 0\,.$$

By Lemma 6.11, for every $i \in [t]$ it holds that $R_a(x_i)$ prints, with high probability, a $\mathcal{TC}^0_{d'}$ circuit of size n^ε whose truth-table is $f(x_i)$. This contradicts the properties of the compression list-refuter. ■

Analogously to Corollary 6.8, we now show that constructing a refuter (rather than a compression-refuter) against probabilistic $\mathcal{TC}^0 \circ \mathsf{SUM}$ circuits suffices for derandomization, and this will induce some overhead in the circuit model. Since now we are concerned with arbitrary functions $f \colon \{0,1\}^n \to \{0,1\}^m$ rather than with $f = \mathsf{Identity}$, we will quantify the output length $m = m(n)$ of f, and account for the overhead in the circuit model according to m.

Corollary 6.12 (refutation implies derandomization for small probabilistic $\mathcal{TC}^0$ circuits)**.** *For every $\varepsilon \in (0,1)$ and $d, d_f, k \in \mathbb{N}_{\geq 1}$ there exist $d' \in \mathbb{N}_{\geq 1}$ and $\delta \in (0,1)$ such that the following holds. Let $m \colon \mathbb{N} \to \mathbb{N}$. Let $f \colon \{0,1\}^n \to \{0,1\}^{m(n)}$ be any function computable by a family of δ-highly uniform threshold circuits of depth d_f and n^k size. Assume the following:*

- *There is a $\mathcal{P}$-computable list-refuter for f against probabilistic*

$$\left(\mathcal{TC}^0_{d'}[(m+n)\cdot n^{\varepsilon}] \mapsto \mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[m\cdot n^{\varepsilon}]\circ n^{\varepsilon}\text{-}\mathsf{SUM}\right)$$

circuits.

Then, there is a deterministic polynomial-time algorithm solving $\mathsf{CAPP}_{0,1/2}$ *for* $\mathcal{TC}^0_d$ *circuits.*

Proof. Let $\varepsilon_1 \in (0,1)$ be a constant to be specified later. We first apply Theorem 6.10 with parameters ε_1, d, d_f, and k, and let d'_1 and δ_1 be the corresponding constants. We let $\delta = \delta_1$. Let $\mu \in \mathbb{N}$ be a sufficiently large constant.

Given the description of a probabilistic $(\mathcal{TC}^0_{d'_1}[n^{1+\varepsilon_1}] \mapsto (\mathcal{TC}^0_{d'_1}\circ\mathsf{SUM})[n^{\varepsilon_1}])$ circuit $\mathcal{C}$, in polynomial-time we can construct the description of a probabilistic

$$\left(\mathcal{TC}^0_{\mu\cdot d'_1}[(m+n)\cdot n^{\mu\cdot\varepsilon_1}] \mapsto (\mathcal{TC}^0_{\mu\cdot d'_1}\text{-}\mathsf{WIRES}[m\cdot n^{\mu\cdot\varepsilon_1}]\circ n^{\varepsilon_1}\text{-}\mathsf{SUM}\right)$$

circuit $\mathcal{C}'$, such that $\mathcal{C}'$ first runs $\mathcal{C}$ and treats its output as the description of a $\mathcal{TC}^0_{d'_1}$ circuit E of n^{ε_1} size, and outputs the first m bits of E's truth-table.[42]

Let $d' = \mu\cdot d'_1$ and $\varepsilon = \mu\cdot\varepsilon_1$. From the above transformation, it follows that a $\mathcal{P}$-computable list-refuter for f against probabilistic $\left(\mathcal{TC}^0_{d'}[m\cdot n^{1+\varepsilon}] \mapsto \mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[m\cdot n^{\varepsilon}]\circ n^{\varepsilon}\text{-}\mathsf{SUM}\right)$-circuits immediately implies a $\mathcal{P}$-computable $(\mathcal{TC}^0_{d'_1}, n^{\varepsilon_1})$-compression list-refuter for f against probabilistic $(\mathcal{TC}^0_{d'_1}[n^{1+\varepsilon_1}] \mapsto (\mathcal{TC}^0_{d'_1}\circ\mathsf{SUM})[n^{\varepsilon_1}])$-circuits. The corollary then follows from Theorem 6.10. ■

We can now prove Theorem 1.4. In the following statement, we use $f\colon \{0,1\}^n \to \{0,1\}^m$ with an arbitrary output length $m = m(n)$; the statement of Theorem 1.4 is obtained by using $m = n^{\varepsilon}$.

Theorem 6.13 (derandomization vs refutation for $\mathcal{TC}^0\circ n^{\varepsilon}$-$\mathsf{SUM}$ circuits)**.** *Let $\varepsilon\in(0,1)$, $m\colon \mathbb{N}\to\mathbb{N}$, and $f\colon\{0,1\}^n\to\{0,1\}^{m(n)}$ be such that*

- *For every $\delta\in(0,1)$, f is computable by a family of δ-highly threshold circuits of constant depth.*
- *For every $d'\in\mathbb{N}$, there is a probabilistic $\mathcal{TC}^0$-computable $1/10$-refuter for f against probabilistic*

$$\left(\mathcal{TC}^0_{d'}[m\cdot n^{1+\varepsilon}] \mapsto \mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[m\cdot n^{\varepsilon}]\circ n^{\varepsilon}\text{-}\mathsf{SUM}\right)\text{-circuits}.$$

Then, for the following three statements, we have $(1) \implies (2) \implies (3)$.

1. *For every $d\in\mathbb{N}$, there is a deterministic polynomial-time algorithm solving* CAPP *for $\mathcal{TC}^0_d$ circuits.*
2. *For every $d'\in\mathbb{N}$, there is a $\mathcal{P}$-computable refuter for f against probabilistic*

$$\left(\mathcal{TC}^0_{d'}[m\cdot n^{1+\varepsilon}] \mapsto \mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[m\cdot n^{\varepsilon}]\circ n^{\varepsilon}\text{-}\mathsf{SUM}\right)\text{-circuits}.$$

[42] More precisely, $\mathbf{C}'$ is the composition of $\mathcal{C}$ and a $\mathcal{TC}_{O(d'_1)}$ circuit U that takes the description of a $\mathcal{TC}^0_{d'_1}$ circuit E of n^{ε_1} size as input, and outputs the first m bits of E's truth-table. It it easy to see that U has $m\cdot n^{O(\varepsilon_1)}$ wires.

3. *For every $d \in \mathbb{N}$, there is a deterministic polynomial-time algorithm solving* $\mathsf{CAPP}_{0,1/2}$ *for* $\mathcal{TC}^0_d$ *circuits.*

Proof. First, note that (2) $\implies$ (3) follows immediately from Corollary 6.12. So it suffices to prove (1) $\implies$ (2).

Fix $d' \in \mathbb{N}$. For convenience, we use $\mathfrak{F}$ to denote probabilistic

$$\left(\mathcal{TC}^0_{d'}[m \cdot n^{1+\varepsilon}] \mapsto \mathcal{TC}^0_{d'}\text{-}\mathsf{WIRES}[m \cdot n^{\varepsilon}] \circ n^{\varepsilon}\text{-}\mathsf{SUM}\right)\text{-circuits.}$$

We first note that given the description of an n-input $\mathfrak{F}$ circuit $\mathcal{C}$, in polynomial time we can construct a $\mathcal{TC}^0$ circuit B such that for every $x \in \{0,1\}^n$, we have $\mathcal{C}(x)$ has the same distribution as $B(x, \mathbf{U}_{r_1})$, where $r_1 \leq \mathrm{poly}(n)$.

Let $\mathcal{R}$ be the probabilistic $\mathcal{TC}^0$ refuter for f. Given the description of an n-input $\mathfrak{F}$ circuit $\mathcal{C}$ as input, with probability at least $9/10$ over its randomness, $\mathcal{R}$ outputs a string $z \in \{0,1\}^n$ such that $\Pr[\mathcal{C}(z) = f(z)] < 1/10$. Now, let r_2 be the number of random bits used by $\mathcal{R}$. We construct the following $\mathcal{TC}^0$ circuit $W \colon \{0,1\}^{r_2} \times \{0,1\}^{r_1} \to \{0,1\}$ as

$$W(\alpha, \beta) = \mathbf{1}\Big[B(\mathcal{R}(\mathcal{C};\alpha), \beta) = f(\mathcal{R}(\mathcal{C};\alpha))\Big].$$

By the condition on $\mathcal{R}$, we know that $\Pr_{\alpha,\beta}[W(\alpha,\beta) = 1] < 1/5$. From (1) and Theorem 3.16, in polynomial deterministic time we can find an $\alpha \in \{0,1\}^{r_2}$ such that $\Pr_{\beta}[W(\alpha,\beta) = 1] < 2/3$, then $\mathcal{R}(\mathcal{C}, \alpha)$ is the output of our deterministic refuter. ■

6.3 Refuting deterministic streaming algorithms vs Lossy Code

In this section we prove Theorem 1.8 and Theorem 1.9.

Reminder of Theorem 1.8. *For any function $f \in \mathcal{FP}$, $\varepsilon \in (0,1)$, a deterministic refuter for f against n^{ε}-space polynomial-time deterministic streaming algorithms implies that* `LossyCode` $\in \mathcal{FP}$.

Proof. The theorem would easily follow from Korten's J-tree construction [Kor22b]. Below we give a much simpler self-contained proof, but the ideas are very similar to Korten's results.

Fix $f \in \mathcal{FP}$ and $\varepsilon \in (0,1)$, and let R be the corresponding refuter from the theorem statement. We show how to solve `LossyCode` $\in \mathcal{FP}$.

Let $C \colon \{0,1\}^n \to \{0,1\}^{n-1}$ and $D \colon \{0,1\}^{n-1} \to \{0,1\}^n$ be two circuits of size s (we have $n \leq s$), interpreted as the input to `LossyCode`. For simplicity, we will assume f_m (the restriction of f on m-bit inputs) is a function from $\{0,1\}^m$ to $\{0,1\}^m$. Since $f \in \mathcal{FP}$, there is a constant $k \in \mathbb{N}$ such that f_m admits an m^k-time single-tape Turing machine. We further assume that that the output of the machine is the first m bits in its tape at the end of the execution.

Let $m = s^{2/\varepsilon}$, we construct the following m^{ε}-space streaming algorithm B that attempts to compute f_m:

- Given streaming access to the input $x \in \{0,1\}^m$, let $\beta = x_{[1,n]}$. For every $i \in \{n+1, \ldots, m^k\}$, we set $\beta \leftarrow C(\beta) \circ x_i$. In other words, we set β as an n-bit *succinct* representation of the string $x \cdot 0^{m^k - m}$, which represent the initial tape of the single-tape Turing machine.[43]

[43] We assume for simplicity that the single-tape Turing machine also gets another input-length tape on which the input length $|x| = m$ is written; so we don't have to include a termination symbol # after x on the input tape.

- Given a string $\beta \in \{0,1\}^n$, consider the string $y \in \{0,1\}^{m^k}$ defined as follows: letting $\beta = z^{(\ell-1)}$, for every i from m^k down to $n+1$, we set $y_i \leftarrow \beta_n$ and $\beta \leftarrow D(\beta_{[1,n-1]})$; and $y_{[n]} \leftarrow \beta$. By its definition, given an index $i \in [m^k]$ and $\beta \in \{0,1\}^n$ as input, one can output y_i using space $O(s)$ and running time $\mathrm{poly}(s) \cdot m^k$. We denote its output by $\mathsf{Access}(\beta, i)$.

  We initialize the location of the head to be $\mathsf{idx} = 1$ and q to be the starting state of Turing machine.

- For every $t \in [m^k]$:

  1. Let $\mathsf{oidx} \leftarrow \mathsf{idx}$. Given $\mathsf{Access}(\beta, \mathsf{oidx})$ and q, get the new content of the oidx-th cell (denoted as $u \in \{0,1\}$) and update idx and q according to the Turing machine.

     Set $\mathsf{tmp} \leftarrow \beta$. Define a string $y \in \{0,1\}^{m^k}$ such that $y_i = \mathsf{Access}(\mathsf{tmp}, i)$ if $i \neq \mathsf{oidx}$, and $y_i = u$ otherwise.
  2. Let $\beta = y_{[1,n]}$. For every $i \in \{n+1, \dots, m^k\}$, we set $\beta \leftarrow C(\beta) \circ y_i$.

- For every $i \in [m]$: output $\mathsf{Access}(\beta, i)$.

Roughly speaking, we use C and D to maintain an n-bit succinct representation β of the current m^k-bit content of the Turing machine tape. The $\mathsf{Access}(\beta, i)$ function allows us to access the i-th bit of the tape in $O(s)$ space and $\mathrm{poly}(m)$ time.[44] The overall running time of B is also bounded by $\mathrm{poly}(m)$.

We use $\beta^{(0)}$ to denote the value of β before the 1-th round (of the execution of the Turing machine) and $\beta^{(t)}$ to denote the value of β at the end of the t-th round. We note that $\beta^{(i)}$ is our succinct representation of the content of the tape after the Turing machine runs for i steps. We also let the string $y^{(t)}$ to denote the string y defined at the t-th round, and $y^{(0)} = x \circ 0^{m^k - m}$.

Now, one can observe that if for every $t \in \{0, 1, \dots, m^k\}$ and for every $j \in [m^k]$, we have $y_j^{(t)} = \mathsf{Access}(\beta^{(t)}, j)$. Then by a simple induction, $y^{(m^k)}$ is the correct tape content at the end of the execution of the Turing machine, meaning that B computes $f(x)$ correctly on input $x \in \{0,1\}^m$.

Hence, running the refuter R on B, we get an input $x \in \{0,1\}^m$ such that $B(x) \neq f_m(x)$, which in particular means there exists t, j such that $y_j^{(t)} \neq \mathsf{Access}(\beta^{(t)}, j)$. By the definition of $\beta^{(t)}$, we can see that in the process of repeatedly applying C on $y^{(t)}$ to obtain $\beta^{(t)}$, at least once we would encounter a β such that $D(C(\beta)) \neq \beta$. This allows us to solve `LossyCode` with input (C, D), and completes the proof. ■

Reminder of Theorem 1.9. *For a function $f \in \{\mathsf{DISJ}, \mathsf{IP}\}$ and $\varepsilon \in (0,1)$, the following are equivalent:*

1. *There is a refuter in $\mathcal{FP}$ for f against n^{ε}-space poly-time deterministic streaming algorithms.*
2. *There is a refuter in $\mathcal{FP}$ for f against $(n-1)$-space poly-time deterministic streaming algorithms.*
3. `LossyCode` $\in \mathcal{FP}$.

[44] The J-tree construction from [Kor22b] allows a much faster access time of $\mathrm{poly}(\log m, s)$; but $\mathrm{poly}(m)$ already suffices for our purpose.

Proof. The (1) ⇒ (3) direction follows immediately from Theorem 1.8. And (2) ⇒ (1) direction is immediate.

In the following we establish the (3) ⇒ (2) direction. We will only show it for DISJ; the proof for IP is almost identical. Given a deterministic $(n-1)$-space n^k-time streaming algorithm B that attempts to solve DISJ_n, we construct an input pair C, D to LossyCode as follows:

1. **The compression circuit** $C\colon \{0,1\}^n \to \{0,1\}^{n-1}$**:** runs B on x as the first half of the input to DISJ, and then output the memory of the algorithm B after reading all of x.
2. **The decompression circuit** $D\colon \{0,1\}^{n-1} \to \{0,1\}^n$**:** Given a memory state $z \in \{0,1\}^{n-1}$, we construct output $x \in \{0,1\}^n$ as follows: for every $i \in [n]$, we run B starting with memory z and the second half being string $e_i \in \{0,1\}^n$ (e_i means only the i-th bit is 1, all others being 0) to obtain an output $\bar{x}$ and set $x_i = \bar{x}$.

Now, since LossyCode $\in \mathcal{FP}$, in polynomial time we can find an input $x \in \{0,1\}^n$ such that $D(C(x)) \neq x$. By definition of C and D, it means that for some $i \in [n]$, B fails on the input (x, e_i). Therefore, we can enumerate all $i \in [n]$ to find out which of the (x, e_i) is the desired counter example. ■

7 Characterization of derandomization via the refuter framework

In this section we explain how using the terminology of refuters allows to capture and generalize previous results. In Section 7.1 we explain how to generalize [LP22a], in Section 7.2 we explain how to generalize [LP22b], and in Section 7.3 we explain how to generalize [CT21].

7.1 Leakage-resilient hardness and refuter for Identity

We first recall the definition of almost-all-input leakage-resilient hardness from [LP22b], and explain why it's equivalent to the existence of refuter for Identity against a certain class of algorithms.

Definition 7.1 (Almost-all-input (a.a.i.) leakage-resilient hardness)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a (multi-output) function. We say that f is almost-all-input (T,ℓ)-leakage resilient hard if for all T-time*[45] *probabilistic algorithms* leak *and A satisfying $\mathsf{leak}(x, f(x)) \le \ell(|x|)$, for all sufficiently long strings x, $A(x, \mathsf{leak}(x, f(x))) \neq f(x)$ with probability at least $2/3$ (over their internal randomness).*

We now define *non-uniform probabilistic one-way efficient communication protocols* (denoted as *one-way efficient CP* for convenience) as a special class of RAM machines: for input length $n \in \mathbb{N}_{\ge 1}$, communication $\ell = \ell(n) \in \mathbb{N}$, and running time $T = T(n) \in \mathbb{N}$, there are two randomized uniform $T(n)$-time algorithms $\mathbb{A}$ and $\mathbb{B}$ that[46] take n-bit input $x \in \{0,1\}^n$ and n-bit advice $a \in \{0,1\}^n$ such that $\mathbb{A}(a, x)$ outputs an ℓ-bit message $m \in \{0,1\}^\ell$ and $\mathbb{B}(a, m)$ outputs a Boolean string.[47] We can also define non-uniform probabilistic efficient communication protocols with

[45] This means the running time of A and leak are bounded by $T(n)$ where $n = |x|$ is the length of their first input.

[46] This means that running time of $\mathbb{A}$ and $\mathbb{B}$ are bounded by $T(n)$.

[47] We fix the advice length to be the same as input length for simplicity, but we can certainly separate them as different parameters. Also, note that here the second agent (modeled by $\mathbb{B}$) has no input.

communication ℓ and running time T in a similar way, by giving the current transcript to $\mathbb{A}$ and $\mathbb{B}$ as an additional input.

We now note that aai leakage-resilient hardness is by definition equivalent to refuter for $\mathsf{Identity}$ against one-way efficient CP.

Observation 7.2. *The following statements are equivalent:*

1. *There is a function $f\colon \{0,1\}^n \to \{0,1\}^n$ that is a.a.i. (T, ℓ)-leakage resilient hard.*

2. *There exists a refuter R for* $\mathsf{Identity}$ *against T-time one-way efficient CP with communication complexity ℓ.*

Proof. From their definitions, there is a one-to-one correspondence between leak and $\mathbb{A}$, A and $\mathbb{B}$, and (crucially) the a.a.i.-leakage resilient hard function f and the refuter R. ■

We can show the following equivalence.

Theorem 7.3. *For every polynomial $T(n) \geq n^{1+\Omega(1)}$, and for every $\varepsilon \in (0,1)$, the following statements are equivalent:*

1. *$pr\mathcal{P} = pr\mathcal{BPP}$.*

2. *There is a $\mathcal{P}$-computable n^{ε}-compression refuter for* $\mathsf{Identity}$ *against probablistic ($\mathsf{SIZE}[n^{1+\varepsilon}] \mapsto \mathsf{SIZE\text{-}XOR}[n^{\varepsilon}]$)-circuits.*

3. *There is a refuter for* $\mathsf{Identity}$ *against T-time one-way efficient CP with communication complexity n^{ε}.*

4. *There is a refuter for* $\mathsf{Identity}$ *against T-time efficient CP with communication complexity $n-1$.*

Proof. It is easy to see that $(4) \implies (3)$. To see that $(3) \implies (2)$, note that a probabilistic ($\mathsf{SIZE}[n^{1+\varepsilon}] \mapsto \mathsf{SIZE\text{-}XOR}[n^{\varepsilon}]$)-circuit $\mathcal{C}$ implies a $n^{1+\varepsilon}$-time one-way efficient CP with communication complexity n^{ε} as follows: $\mathbb{A}(x)$ simulates $\mathcal{C}(x)$, and sends its n^{ε}-bit output ℓ to $\mathbb{B}$. $\mathbb{B}(x, \ell)$ treats ℓ as an $\log n$-input n^{ε}-size circuit and outputs its truth-table.

We note that $(2) \implies (1)$ follows from an identical proof as in Theorem 6.7. To show $(1) \implies (4)$, we note that for any T-time efficient CP $\mathcal{P} = (\mathbb{A}, \mathbb{B})$ with communication complexity at most $1/2$, we have $\Pr_{z \in \{0,1\}^n}[\mathcal{P}(z) = z] \leq 1/2$ (the randomness is also over the inner randomness of $\mathcal{P}$) by a simple counting argument. Assuming $pr\mathcal{P} = pr\mathcal{BPP}$ and applying Theorem 3.16, we can find an z such that $\Pr_{[}\mathcal{P}(z) = z] < 2/3$ deterministically. This completes the proof. ■

Remark 7.4. *We remark that Item (2) in Theorem 7.3 is indeed (syntactically) equivalent to the notion of a.a.i. leakage-resilient hardness* local *hardness in [LP22b].*

In particular, the equivalence between Item (1) and Item (2) above shows that even assuming the leak function from Definition 7.1 to be a probabilistic $\mathsf{SIZE} \circ \mathsf{XOR}[n^{\varepsilon}]$ circuit sampled by an $n^{1+\varepsilon}$ circuit and the A function to be the truth-table generation function (given an n^{ε}-size circuit, output its length-n truth-table) that does not depend on the input x, the existence of a.a.i. leakage resilient hard is still equivalent to derandomization.

7.2 Hardness of Conditional Kolmogorov Complexity

We now explain how the viewpoint of refuters allows to generalize the results of [LP22a]. To do so, let us first recall the definitions of Levin's Kolmogorov complexity and of the problem $\mathsf{GapMcKtP}$, which refes to *conditional* Levin's Kolmogorov complexity.

Definition 7.5 (Levin's Kolmogorov complexity)**.** *For a fixed universal Turing machine U, and any $x, z \in \{0,1\}^*$, we define*

$$Kt(x|z) = \min_{\Pi \in \{0,1\}^*, t \in \mathbb{N}} \left\{ |\Pi| + \log(t) : U(\Pi(z), 1^t) = x \right\} .$$

Definition 7.6 ($\mathsf{GapMcKtP}$)**.** *Let $T_{\mathsf{YES}}, T_{\mathsf{NO}} \colon \mathbb{N} \to \mathbb{N}$. The problem problem $\mathsf{GapMcKtP}[T_{\mathsf{YES}}, T_{\mathsf{NO}}]$ is defined as follows:*

- YES *instances: (x,z) such that $|x| = |z|$ and $Kt(x|z) \le T_{\mathsf{YES}}(|x|)$.*
- NO *instances: (x,z) such that $|x| = |z|$ and $Kt(x|z) \ge T_{\mathsf{NO}}(|x|)$.*

The main result from [LP22a] asserts that derandomization is equivalent to hardness of $\mathsf{GapMcKtP}$ against probabilistic polynomial-time algorithms on *almost all conditions z*; that is, for every algorithm and every z (except, at most, finitely many), there is an x such that the algorithm fails on input (x,z).

Theorem 7.7 (derandomization vs almost-all-conditions hardness of $\mathsf{GapMcKtP}$; [LP22a, Theorem 1])**.** *There exists a constant $c \ge 1$ such that the following two statements are equivalent.*

1. *$pr\mathcal{BPP} = pr\mathcal{P}$.*
2. *There exists $\gamma \in \mathbb{R}$ such that for every probabilistic algorithm M running in time n^c, for all but finitely many $z \in \{0,1\}^*$, there exists $x \in \{0,1\}^*$ such that M fails to solve $\mathsf{GapMcKtP}[\gamma \cdot \log(n), n-1]$ correctly on input (x,z).*

We now show that Corollary 6.5 is a strengthening of Theorem 7.7. Specifically, we prove that the hypothesis of Theorem 7.7 is at least as strong as the hypothesis in Corollary 6.5, which asserts the existence of a compression list-refuter for probabilistic algorithms running in time n^c.

Claim 7.8 (hardness of $\mathsf{GapMcKtP}$ implies refutation)**.** *Suppose that the hypothesis in Item (2) of Theorem 7.7 holds. Then, there exists a $\mathcal{P}$-computable $\sqrt{n}$-compression list-refuter for $\mathsf{Identity}$ against general probabilistic algorithms running in time $n^{c-o(1)}$.*

Proof. The refuter Ref gets input (M,a), where $|a| = n$, and enumerates over all strings $\Pi_1, ..., \Pi_{2^{\ell+1}-1}$ of length at most $\ell = \gamma \cdot \log(n)$. Treating each Π_i as the description of a RAM, it simulates the machine for 2^ℓ steps on input a, and if the machine prints an n-bit string w_i, then Ref prints w_i (otherwise, the refuter just moves on to Π_{i+1}). The final output list of Ref consists of all w_i's that it printed.

Assume towards a contradiction that there is a time-$n^{c-o(1)}$ RAM M' and an infinite set $A \subseteq \{0,1\}^*$ such that for every $a \in \{0,1\}^*$, for all w_i that $\mathsf{Ref}(M',a)$ prints, it holds that

$$\Pr\left[M'(a, w_i) \text{ prints a circuit of size } \sqrt{2|a|} \text{ whose truth-table is } w_i \right] \ge 2/3 ,$$

where the probability is over the random coins of M'.[48]

Then, for any $z \in A$, for all x we solve $\mathsf{GapMcKtP}[\gamma \cdot \log(n), n-1]$ on input (x,z) as follows. Given (x,z), we simulate $M'(z,x)$ for constantly many independent trials; if in one of those trials, M outputs a circuit of size $\sqrt{2|z|}$ with truth-table equal to x then we accept, otherwise we reject. Note that on "no" instances, we always reject (because no such circuit exists). On "yes" instances, by definition there exists a program Π of size at most ℓ running in at most 2^ℓ steps such that $\Pi(z) = x$. By the definition of Ref, one of the outputs in the list that $\mathsf{Ref}(M', z)$ prints will be x. By the assumption on M' and the fact that $a \in Z$, with high probability $M'(z,x)$ prints a circuit of size $\sqrt{2|z|}$ with truth-table x, therefore we accept. ■

7.3 Almost-all-inputs hardness

We now explain how the viewpoint of refuters also allows us to capture and generalize the results of Chen and Tell [CT21]. Recall that they considered the notion of hardness on almost all inputs, defined as follows:

Definition 7.9 (almost-all-inputs hardness)**.** *A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is hard on* almost all inputs *for probabilistic algorithms running in time T if for every T-time algorithm M and for all but finitely many inputs x, $\Pr[M(x) = f(x)] < 2/3$.*

The main result of [CT21] is a two-way connections between derandomization (i.e., $pr\mathcal{BPP} = pr\mathcal{P}$) and the existence of functions that are hard on almost all inputs for probabilistic algorithms running in fixed polynomial time.

Theorem 7.10 (the main result of [CT21])**.** *For any $\ell = \mathrm{polylog}(n)$, the following statements hold:*

1. *If there is a function mapping n bits to n/ℓ bits that is computable by logspace-uniform circuits of polynomial size and depth $O(n^2)$, and that is hard for probabilistic time n^c on almost all inputs, where $c > 1$ is a sufficiently large universal constant, then $pr\mathcal{BPP} = pr\mathcal{P}$.*
2. *If $pr\mathcal{BPP} = pr\mathcal{P}$, then for every $c \geq 1$ there is a function in $\mathcal{FP}$ mapping n bits to n/ℓ bits that is hard for probabilistic time n^c on almost all inputs.*

The original statement in [CT21] referred to length-preserving functions, but (as mentioned in that paper) the precise output length is immaterial for the result. We have chosen to present the result in Theorem 7.10 using output length $n/\mathrm{polylog}(n)$ to facilitate capturing cleanly it using refuter terminology.

To capture Theorem 7.10 in refuter terminology, we define algorithms that get advice and do not examine their input as the class of RAMs M that get two inputs and satisfy the following: for every $a \in \{0,1\}^*$ and every $x, x' \in \{0,1\}^*$ such that $|x| = |x'|$ it holds that $M(x,a) = M(x',a)$. (When M is probabilistic, we require the equality to hold for every fixed choice of random coins.)

The following claim asserts that hardness on almost all inputs is equivalent to refuting algorithms that do not examine their input.

Claim 7.11 (almost-all-inputs hardness is equivalent to refuting machines that don't examine their inputs)**.** *For any polynomial $T(n) \geq n^2$, the following statements are equivalent:*

[48] The reason that the circuit size is $\sqrt{2|a|}$ instead of $\sqrt{|a|}$ is that we defined s-compression refuters with $s = \sqrt{\cdot}$ being a function of $|a| + |w_i| = 2|a|$.

1. *There is an $\mathcal{FP}$-refuter for* Identity *against algorithms that on n-bit inputs run in probabilistic time $O(T(\tilde{O}(n)))$, get $\tilde{O}(n)$ bits of advice, and do not examine their input.*

2. *There is a function $f \in \mathcal{FP}$ mapping n bits to $n/\mathrm{polylog}(n)$ bits that is hard on almost all inputs for probabilistic algorithms running in time $O(T)$.*

Proof. We first prove that (1) $\Rightarrow$ (2). Let R be the refuter, let $\ell(n) = \mathrm{polylog}(n)$ be a sufficiently large polylogarithm. Given input $x \in \{0,1\}^n$, consider the first $m = \log(n)$ Turing machines, denoted $M_1, ..., M_m$, according to some canonical enumeration. For every $i \in [m]$, we compute $y_i = R(M_i, x)) \in \{0,1\}^{n/\ell}$,[49] and print the string

$$f(x) = y_1 \circ y_2 \circ ... \circ y_m \ ,$$

which is of length $n/\ell \cdot \log(n) = n/\mathrm{polylog}(n)$. (For $i \in [m]$ such that the refuter does not output a string y_i, we print $y_i = 0^{n/\ell}$.)

Assume towards a contradiction that there is a time-$O(T)$ Turing machine F and an infinite set $X \subseteq \{0,1\}^*$ such that for every $x \in X$ it holds that $\Pr[F(x) = f(x)] \geq 2/3$. Let A be an advice-taking machine that on any input of length n, and given advice $x \in \{0,1\}^N$ where N satisfies $n = N/\ell$, simulates F on input x and outputs the $(i_A)^{th}$ substring of $F(x)$, where i_A is A's index in the enumeration of Turing machines.[50] Note that the advice complexity of A is $N = n \cdot \ell(N) = \tilde{O}(n)$, and its running time is $O(T(N)) = O(T(\tilde{O}(n)))$. Thus, for every sufficiently long $x \in X$ we have

$$\begin{aligned} \Pr\left[A\Big(R(A,x), x\Big) = R(A,x)\right] &= \Pr[F(x)_{i_A} = R(A,x)] \\ &\geq \Pr[F(x) = f(x)] \\ &\geq 2/3 \ , \end{aligned}$$

which contradicts the properties of the refuter.

Now, let us prove that (2) $\Rightarrow$ (1). For a sufficiently large polylogarithm $\ell = \mathrm{polylog}(n)$, the refuter gets input (M, a) where M is the description of a T-time machine that does not examine its input and $a \in \{0,1\}^n$, and the refuter outputs $f(a) \in \{0,1\}^{n/\ell}$. Assume towards a contradiction that for some machine M running in time $T'(m) = O(T(\tilde{O}(m)))$ and infinitely many advice strings $a \in \{0,1\}^*$ it holds that $\Pr[M(f(a), a) = f(a)] \geq 2/3$. Consider the machine M' that gets input $a \in \{0,1\}^n$ and outputs $M(0^{n/\ell}, a)$. The running time of M' is $T'(n/\ell) < O(T(n))$, and we have that $M'(a) = M(0^{n/\ell}, a) = M(f(a), a)$. Thus, for infinitely many a's we have that $\Pr[M'(a) = f(a)] \geq 2/3$, a contradiction. ■

Observe that in the proof above, the refuter and the almost-all-inputs hard function have essentially the same complexity. In particular, if one is computable by logspace-uniform circuits of polynomial size and depth n^2, then the other is computable by logspace-uniform circuits of polynomial size and depth $O(n^2)$. Hence, we can present Theorem 7.10 in refuter terminology:

Corollary 7.12 (the main result of [CT21], in refuter terminology)**.** *For every $c \geq 1$, let $\mathcal{O}_c$ be the class of probabilistic algorithms that on n-bit inputs run in time n^c, get $\tilde{O}(n)$ bits of advice, and do not examine their input. Then, the following statements hold:*

[49] We ignore rounding issues throughout the proof, for simplicity.

[50] We can assume that A can use its own index i_A in its execution, by Kleene's recursion theorem and assuming an efficient mapping of machine descriptions to their indices in the enumeration of machines.

1. *For a sufficiently large $c \geq 1$, assume that there is a refuter for* Identity *against $\mathcal{O}_c$ that is computable by logspace-uniform circuits of polynomial size and depth n^2. Then, $pr\mathcal{BPP} = pr\mathcal{P}$.*

2. *If $pr\mathcal{BPP} = pr\mathcal{P}$, then for every constant $c > 1$ there is an $\mathcal{FP}$-refuter for* Identity *against $\mathcal{O}_c$.*

Acknowledgments

Part of this work was done while the authors were visiting the Simons Institute for the Theory of Computing. We are grateful to anonymous FOCS reviewers for pointing out various typos and inaccuracies.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, Cambridge, 2009.

[ABN+92] N. Alon, J. Bruck, J. Naor, M. Naor, and R. M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, 1992.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. volume 58, pages 137–147. 1999. Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996).

[Bea91] Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM Journal of Computing*, 20(2):270–277, 1991.

[BF99] Harry Buhrman and Lance Fortnow. One-sided versus two-sided error in probabilistic computation. In *Proc. 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 100–109, 1999.

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.

[CJSW21] Lijie Chen, Ce Jin, Rahul Santhanam, and Ryan Williams. Constructive separations and their consequences. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 646–657, 2021.

[CT21] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2021.

[DT23] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. *Electronic Colloquium on Computational Complexity: ECCC*, 30:036, 2023.

[GG79] Ofer Gabber and Zvi Galil. Explicit constructions of linear size superconcentrators. In *Proc. 20th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 364–370. 1979.

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

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, 2015.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

[Gol11] Oded Goldreich. In a world of P=BPP. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*, pages 191–232, 2011.

[GSTS07] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. If NP languages are hard on the worst-case, then it is easy to find their hard instances. *Computational Complexity*, 16(4):412–441, 2007.

[GV04] Dan Gutfreund and Emanuele Viola. Fooling parity tests with parity gates. In *Proc. 8th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 381–392, 2004.

[GZ11] Oded Goldreich and David Zuckerman. Another proof that $\mathsf{BPP} \subseteq \mathsf{PH}$ (and more). In *Studies in complexity and cryptography*, volume 6650 of *Lecture Notes in Comput. Sci.*, pages 40–53. Springer, Heidelberg, 2011.

[HAB02a] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002.

[HAB02b] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002.

[IW97] Russell Impagliazzo and Avi Wigderson. $\mathrm{P} = \mathrm{BPP}$ if E requires exponential circuits: derandomizing the XOR lemma. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[IW98] Russell Impagliazzo and Avi Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 734–743, 1998.

[Kab01] Valentine Kabanets. Easiness assumptions and hardness tests: trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001.

[Kor22a] Oliver Korten. Derandomization from time-space tradeoffs. In *Proc. 37th Annual IEEE Conference on Computational Complexity (CCC)*, volume 234 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 37, 26. 2022.

[Kor22b] Oliver Korten. Derandomization from time-space tradeoffs. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPIcs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[LP22a] Yanyi Liu and Rafael Pass. Characterizing derandomization through hardness of Levin-Kolmogorov complexity. In *Proc. 37th Annual IEEE Conference on Computational Complexity (CCC)*, volume 234 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 35, 17. 2022.

[LP22b] Yanyi Liu and Rafael Pass. Leakage-resilient hardness v.s. randomness. *Electronic Colloquium on Computational Complexity: ECCC*, TR22-113, 2022.

[MW18] Dylan M. McKay and Richard Ryan Williams. Quadratic Time-Space Lower Bounds for Computing Natural Functions with a Random Oracle. In *Proc. 10th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 56:1–56:20, 2018.

[Nis93] Noam Nisan. The communication complexity of threshold gates. In *Combinatorics, Paul Erdős is eighty, Vol. 1*, Bolyai Society Mathematical Studies, pages 301–315. 1993.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[Raz92] A. A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

[Sip83] Michael Sipser. A complexity theoretic approach to randomness. In *Proc. 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 330–335, 1983.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[Sud97] Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *J. Complex.*, 13(1):180–193, 1997.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *Journal of Computer and System Sciences*, 67(2):419–440, 2003.

[Vad12] Salil P. Vadhan. *Pseudorandomness*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2012.

[VL99] Jacobus Hendricus Van Lint. *Introduction to coding theory*, volume 86. Springer-Verlag Berlin Heidelberg, 1999.

[vMS23] Dieter van Melkebeek and Nicollas Sdroievski. Instance-wise hardness versus randomness tradeoffs for arthur-merlin protocols. *Electronic Colloquium on Computational Complexity: ECCC*, 30:029, 2023.

A The tarHSG of [CT21] with low-space streaming reconstruction

In this section we prove Theorem 3.15 (restated below).

Reminder of Theorem 3.15. *There exists a universal constant $c > 1$ such that the following holds. Let $f\colon \{0,1\}^N \to \{0,1\}^N$ be computable in time $T(N)$, let $\gamma > 0$, and let $M\colon \mathbb{N} \to \mathbb{N}$ such that $c \cdot \log(T) < M < T^{\gamma/c}$. Then, there exists a deterministic algorithm $H_f^{\mathtt{CT}}$ and a probabilistic oracle machine $R_f^{\mathtt{CT}}$ that for every $z \in \{0,1\}^N$ satisfy the following:*

1. **Generator:** *When given input z, the machine $H_f^{\mathtt{CT}}$ runs in time $\mathrm{poly}(T(N))$ and prints a list of strings in $\{0,1\}^M$.*
2. **Reconstruction:** *$R_f^{\mathtt{CT}}$ gets input z, and can be implemented by a T^γ-space one-pass streaming algorithm over the input z with running time $M^c \cdot T^{1+\gamma}$. When $R_f^{\mathtt{CT}}$ is given oracle access to a function $D\colon \{0,1\}^M \to \{0,1\}$ that $1/M$-avoids $H_f^{\mathtt{CT}}(z)$, with probability at least $1 - 1/M$ the machine $R_f^{\mathtt{CT}}$ outputs an oracle circuit $C_{f(z)}$ of size T^γ such that the truth-table of $(C_{f(z)})^D$ is $f(z)$.*

Proof sketch. From our assumption, f is also computable by a logspace-uniform circuit of $\widetilde{O}(T)$ size and $\widetilde{O}(T)$ depth. We follow the reconstruction algorithm described in [CT21, Section 4.4] and observe that everything except for the first iteration takes only

$$(t \cdot T^\gamma \cdot M)^{4c_0^2} \cdot (d + N) \le T^{1+O(\gamma)} \cdot M^{O(1)}$$

time and $T^{O(\gamma)}$ space. Moreover, the first iteration is the only place where the algorithm needs access to the input string z.

Hence, the remaining challenge is to implement the first phase by a $T^{O(\gamma)}$-space one-pass streaming algorithm. The original reconstruction algorithm in [CT21, Section 4.4] constructs a circuit of size $t_0 \ge N$ for the first polynomial p_1, which already requires N bits to restore (which can be much larger than the $T^{O(\gamma)}$ space bound we aim for). We observe that this is not necessary: instead of building a circuit C_1 for p_1, we can directly start from building a circuit C_2 for p_2, and using the t_0-time base case algorithm to answer all queries when running [CT21, Lemma 4.10] for $i = 2$. This can be done in $T^{O(\gamma)} \cdot \mathrm{poly}(M) \cdot t_0 \le T^{1+O(\gamma)} \cdot \mathrm{poly}(M)$ time and only uses $T^{O(\gamma)}$ space.

Moreover, we can further observe that [CT21, Lemma 4.10] only makes $T^{O(\gamma)}$ non-adaptive queries to C_{i-1}, meaning that one can first gather all these queries using $T^{O(\gamma)}$ space, and then try to answer all of them together using a single pass over the input. We note that p_1 correspond to the input polynomial $\hat{\alpha}_0\colon \mathbb{F}^m \to \mathbb{F}$, which is defined by

$$\hat{\alpha}_0(\vec{w}) = \sum_{\vec{z} \in H^{m'} \times \{0\}^{m-m'}} \delta_{\vec{z}}(\vec{w}) \cdot \alpha_0(\vec{z}) \ ,$$

where $\delta_{\vec{z}}$ is Kronecker's delta function (i.e., $\delta_{\vec{z}}(\vec{w}) = \prod_{j \in [m]} \prod_{a \in H \setminus \{z_j\}} \frac{w_j - a}{z_j - a}$) and $\alpha_0(\vec{z})$ denotes an input bit to f indexed by $\vec{z}$. From its definition, one can see that in $O(\log T)$ space one can compute $\hat{\alpha}_0(\vec{w})$ via a single pass over the input.

Finally, we can set the γ above small enough compared to the γ in the statement, and the whole algorithm can be implemented by a T^γ-space one-pass streaming algorithm over the input z with running time $M^c \cdot T^{1+\gamma}$. This completes the proof. ∎

B The STV PRG with $\mathcal{TC}^0 \circ \mathsf{XOR}$ reconstruction

B.1 Finite Fields

Throughout this section, we will only consider finite fields of the form $\mathsf{GF}(2^{2\cdot 3^\ell})$ for some $\ell \in \mathbb{N}$ since they enjoy simple representations that will be useful for us. We say $p = 2^r$ is a *nice power* of 2, if $r = 2 \cdot 3^\ell$ for some $\ell \in \mathbb{N}$.

Let $\ell \in \mathbb{N}$ and $n = 2 \cdot 3^\ell$. In the following we use $\mathbb{F}$ to denote $\mathbb{F}_{2^n}$ for convenience. We will always represent GF_{2^n} as $\mathbb{F}_2[\mathbf{x}]/(\mathbf{x}^n + \mathbf{x}^{n/2} + 1)$.[51] That is, we identify each element of $\mathsf{GF}(2^n)$ with an $\mathbb{F}_2[\mathbf{x}]$ polynomial of degree less than n. To avoid confusion, given a polynomial $P(\mathbf{x}) \in \mathbb{F}_2[\mathbf{x}]$ with degree less than n, we will use $(P(\mathbf{x}))_{\mathbb{F}}$ to denote the unique element in $\mathbb{F}$ identified with $P(\mathbf{x})$.

Let $\kappa^{(n)}$ be the natural bijection between $\{0,1\}^n$ and $\mathbb{F} = \mathsf{GF}(2^n)$: for every $a \in \{0,1\}^n$, $\kappa^{(n)}(a) = \left(\sum_{i\in[n]} a_i \cdot \mathbf{x}^{i-1}\right)_{\mathbb{F}}$. We always use $\kappa^{(n)}$ to encode elements from $\mathbb{F}$ by Boolean strings. That is, whenever we say that an algorithm takes an input from $\mathbb{F}$, we mean it takes a string $x \in \{0,1\}^n$ and interprets it as an element of $\mathbb{F}$ via $\kappa^{(n)}$. Similarly, whenever we say that an algorithm outputs an element from $\mathbb{F}$, we mean it outputs a string $\{0,1\}^n$ encoding that element via $\kappa^{(n)}$. For simplicity, sometimes we use $(a)_{\mathbb{F}}$ to denote $\kappa^{(n)}(a)$. Also, when we say the i-th element in $\mathbb{F}$, we mean the element in $\mathbb{F}$ encoded by the i-th lexicographically smallest Boolean string in $\{0,1\}^n$.

B.2 Proof of Theorem 3.14

Theorem B.1 (the STV PRG with $\mathcal{TC}^0 \circ \mathsf{XOR}$ reconstruction)**.** *There are universal constants $c_{\mathsf{STV}} > 1$ and $d_{\mathsf{STV}} \in \mathbb{N}_{\geq 1}$ such that for every sufficiently small constant $\bar{\gamma} \in (0,1)$, there are deterministic algorithms G^{STV} and R^{STV} that satisfy the following:*

1. **Generator:** *When given a string $z \in \{0,1\}^n$, G^{STV} runs in time $\mathrm{poly}(n)$ and prints a list of strings in $\{0,1\}^m$, where $m = n^{\bar{\gamma}}$.*

2. **Reconstruction:** *$R^{\mathsf{STV}}(1^n)$ outputs the description of a probabilistic*

$$\left(\mathcal{TC}^0_{d_{\mathsf{STV}}}[n \cdot m^{c_{\mathsf{STV}}}] \mapsto \mathcal{TC}^0_{d_{\mathsf{STV}}} \circ \mathsf{XOR}[m^{c_{\mathsf{STV}}}]\right)$$

oracle circuit $\mathcal{R}_f$, such that given $D\colon \{0,1\}^m \to \{0,1\}$ that $1/m$-distinguishes $G^{\mathsf{STV}}(z)$ as oracle, we have

$$\Pr_{R_f \leftarrow \mathcal{R}_f}\left[R_f^D(z) \text{ outputs a } \mathcal{TC}^0_{d_1} \text{ oracle circuit } E \text{ such that } \mathsf{tt}(E^D) = z\right] \geq 2/3.$$

Proof. We begin by setting some notation.

Notation. Let h be the smallest nice power of 2 that is at least m. Let $p = h^{27}$ (therefore p is also a nice power of 2). Let ℓ be the smallest integer such that $h^\ell \geq n$. Let $\mathbb{F} = \mathbb{F}_p$ and H be the

[51] Note that $\mathbf{x}^{2\cdot 3^\ell} + \mathbf{x}^{3^\ell} + 1 \in \mathbb{F}_2[\mathbf{x}]$ is always irreducible; see [VL99, Theorem 1.1.28].

first h elements from $\mathbb{F}_p$. Let $\zeta\colon [n] \to H^m$ be an efficiently computable injection mapping.[52] Let $z \in \{0,1\}^n$ be our input. Let $c_{\texttt{NW}}$ and $d_{\texttt{NW}}$ be the universal constants from Theorem 3.13.

Let $d_0 \in \mathbb{N}$ be a sufficiently large constant such that $d_0 \geq d_{\texttt{NW}}$. Let $\mu \in \mathbb{N}$ be a sufficiently large constant.

The generator $G^{\texttt{STV}}$. First, we define $P_z\colon \mathbb{F}^\ell \to \mathbb{F}$ as

$$P_z(\vec{u}) = \sum_{i\in[n],\ \vec{w}=\zeta(i)} \delta_{\vec{w}}(\vec{u}) \cdot a_i\ ,$$

where $\delta_{\vec{w}}$ is Kronecker's delta function (i.e., $\delta_{\vec{w}}(\vec{u}) = \prod_{j\in[\ell]} \prod_{a\in H\setminus\{z_j\}} \frac{u_j-a}{w_j-a}$). Let $d = \ell\cdot(h-1)$ be the degree of P_z.

From our choice of h, we know that $m \leq h \leq m^3$. We also have $n \leq h^\ell \leq n^2$, and $n^{27} \leq p^\ell \leq n^{54}$.

Let $\hat{z} = \mathsf{tt}(P_z) \in \mathbb{F}^{|\mathbb{F}|^\ell}$ and let $N = |\hat{z}| = |\mathbb{F}|^\ell$. We instantiate Theorem 4.1 with $\gamma = \bar{\gamma}$ and $\nu = \bar{\gamma}$. Note that $N^{c_0\cdot(\gamma+\nu)} \leq \mathsf{poly}(m)$. Let c_0 be the universal constant from Theorem 4.1 and $c^\star = c^\star_{\gamma,\nu}$ be the corresponding constant. Let $\bar{z} = \mathsf{Enc}(\hat{z})$ and $\bar{N} = |\bar{z}|$. Note that $\bar{N} = N^{c^\star}$. Now let γ_1 so that $N^{c^\star\cdot\gamma_1} = n^{\bar{\gamma}} = m$ (note that γ_1 is not a constant, but since $N \leq \mathsf{poly}(n)$ by the definition of h, p, we have that γ_1 is bounded away from 0), and we define

$$G^{\texttt{STV}}(z) = G^{\texttt{NW}}(\bar{z}, m).$$

Note that $G^{\texttt{STV}}(z)$ runs in $\mathsf{poly}(n)$ time as desired.

Reconstruction $R^{\texttt{STV}}$. We need the following fact.

Fact B.2. *The following two statements hold:*

1. *There is a $\mathcal{P}$-uniform $n\cdot\mathsf{poly}(m)$-size $\mathcal{TC}^0_{d_0}$ circuit that takes input $i \in [|\hat{z}|]$ and outputs a circuit G_i consisting of $(\log_2 p)$ XOR gates such that $G_i(z) = \hat{z}_i$ for all $z \in \{0,1\}^n$.*

2. *There is a $\mathcal{P}$-uniform $n\cdot\mathsf{poly}(m)$-size $\mathcal{TC}^0_{d_0}$ circuit that takes input $i \in [|\bar{z}|]$ and outputs a $\mathsf{poly}(m)$-size $\mathcal{TC}^0_{d_0} \circ \mathsf{XOR}$ circuit W_i such that $W_i(z) = \bar{z}_i$ for all $z \in \{0,1\}^n$.*

Proof. Let $i \in [\hat{z}]$ and $\vec{w} \in \mathbb{F}^\ell$ be the corresponding vector. To compute the gate G_i, it suffices to compute the coefficients $\beta_k = \delta_{\zeta(k)}(\vec{w})$ for every $k \in [n]$ (so that $\hat{z}_i = P_z(\vec{w}) = \sum_{k\in[n]} \beta_k \cdot a_k$). From the definition of $\delta_{\zeta(k)}(\vec{w})$, this can be by a $\mathcal{P}$-uniform $n\cdot\mathsf{poly}(m)$-size $\mathcal{TC}^0_{d_0}$ circuit.

The circuit W_j is computed as follows:

1. Given input $i \in [|\bar{z}|]$. Run $Q_N(i)$ to obtain a list $q_1, \ldots, q_M \in [N]$, where $M = N^\gamma$.

2. For each $j \in [M]$, interpreting q_j as a vector $\vec{w}_j \in \mathbb{F}^\ell$. Output the circuit W_i defined as

$$W_i(z) = E_N(i, G_{q_i}(z), \ldots, G_{q_M}(z)).$$

Note that $|Q_N|, |E_N| \leq N^{c_0\cdot(\gamma+\nu)} \leq \mathsf{poly}(m)$. Hence, W_j can be computed from j by a $\mathcal{P}$-uniform $n\cdot\mathsf{poly}(m)$-size $\mathcal{TC}^0_{d_0}$. ∎

[52] For simplicity we ignore the complexity of computing ζ since it is negligible.

Let $S_{\mathtt{NW}} = R^{\mathtt{NW}}(1^{|\bar{z}|}, m)$. Without loss of generality, we assume that $S_{\mathtt{NW}}$ takes exactly $r_{\mathtt{NW}} = m^{c_{\mathtt{NW}}}$ bits as input.

In the following we will construct two samplers S_1 and S_2, and combine them to obtain our final sampler S.

Claim B.3. *There is a polynomial-time algorithm that, given 1^n, outputs a $\mathcal{TC}^0_{O(d_0)}[n \cdot m^{c_{\mathtt{STV}}/2}]$ circuit S_1 satisfying the following:*

1. *S_1 takes $r_1 = r_{\mathtt{NW}}$ bits as input, and outputs the description of a $\mathrm{poly}(m)$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{XOR}$ circuit E_1.*

2. *E_1 takes $z \in \{0,1\}^n$ as input, and outputs the description of a $m^{c_{\mathtt{NW}}}$-size $\mathcal{TC}^0_{d_{\mathtt{NW}}}$ oracle circuit C_1.*

3. *For every $z \in \{0,1\}^n$, with probability at least 0.99 over $E_1 \leftarrow S_1(\mathbf{U}_{r_1})$, letting $C_1 = E_1(z)$, it holds that*
$$\Pr_{i \in [\bar{N}]}\left[C_1^D(i) = \bar{z}_i\right] \geq 1/2 + m^{-3}.$$

Proof. Formally, given $\alpha_1 \in \{0,1\}^{r_1}$, S_1 computes all the queries of $S_{\mathtt{NW}}$ made to $\bar{a}$ in $\mathcal{TC}^0_{d_{\mathtt{NW}}}[m^{c_{\mathtt{NW}}}]$ (note that $S_{\mathtt{NW}}$ is a non-adaptive oracle circuit), and applies Fact B.2 to replace all calls to $\bar{a}$ in $S_{\mathtt{NW}}$ by $\mathrm{poly}(m)$-size $\mathcal{TC}^0_{d_0} \circ \mathsf{XOR}$ circuits with input $z \in \{0,1\}^n$. This way, S_1 outputs the desired $\mathrm{poly}(m)$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{XOR}$ circuit E_1.

Moreover, by Fact B.2, we know that S_1 can be implemented by a $\mathcal{TC}^0_{O(d_0)}$ circuit of $n \cdot \mathrm{poly}(m)$ size. ∎

Claim B.4. *There is a polynomial-time algorithm that, given 1^n, outputs a $\mathcal{TC}^0_{O(d_0)}[n \cdot m^{c_{\mathtt{STV}}/2}]$ circuit S_2 satisfying the following:*

1. *S_2 takes $r_2 = m^{c_{\mathtt{STV}}/2}$ bits as input, and outputs the description of a $\mathrm{poly}(m)$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{XOR}$ circuit E_2.*

2. *E_2 takes $z \in \{0,1\}^n$ as input, and outputs the description of a m^{μ}-size $\mathcal{TC}^0_{d_0}$ oracle circuit C_2.*

3. *For every $z \in \{0,1\}^n$ and every oracle $O\colon [\bar{N}] \to \{0,1\}$ such that $\Pr_{i \in [\bar{N}]}[O(i) = \bar{z}_i] \geq 1/2 + m^{-3}$, with probability at least $1 - 0.99$ over $E_2 \leftarrow S_2(\mathbf{U}_{r_2})$, letting $C_2 = E_2(z)$, it holds that*
$$\Pr_{i \in [\bar{N}]}\left[C_2^O(i) = \bar{z}_i\right] \geq 1 - 1/d^2.$$

Proof. Let $r_{\mathsf{pre}}, r_{\mathsf{main}} \leq T^{c_1 \cdot \delta}$ be the number of random bits used by D_N of Proposition 5.5 for the preprocessing step and the main step, respectively. (We use the main step to denote the operation of REC_n after the preprocessing step.)

Let S_{pre} and S_{main} be the $\mathcal{TC}^0_{d_0}[N^{c_0 \cdot (\eta + \nu)}]$ samplers for the preprocessing step and the main step of D_N, respectively. In more detail: (1) S_{pre} takes $\alpha_{\mathsf{pre}} \in \{0,1\}^{r_{\mathsf{pre}}}$ bits as input, and outputs a list of queries to $\hat{z}$, denoted by $q_1, q_2, \ldots, q_t \in [t]$, where $t \leq N^{c_0 \cdot (\eta + \nu)}$; (2) S_{main} takes $\alpha_{\mathsf{main}} \in \{0,1\}^{r_{\mathsf{main}}}$ as input, and outputs a $\mathcal{TC}^0_{d_0}$ oracle circuit C_2' of size $N^{c_0 \cdot (\eta + \nu)}$ that takes t bits and $j \in [N]$ as input.

The promise of Proposition 4.1 implies that for any $O\colon \{0,1\}^{\bar{N}} \to \{0,1\}$ satisfying $\Pr_{j \in [\bar{N}]}[O(j) = \bar{z}(j)] \geq 1/2 + N^{-\nu}$, with probability at least $1 - o(1)$ over $\alpha_{\mathsf{pre}} \leftarrow U_{r_{\mathsf{pre}}}$ and $\alpha_{\mathsf{main}} \leftarrow U_{r_{\mathsf{main}}}$, it holds that $C_2^O(j) := (C_2')^O(\hat{z}_{q_1}, \ldots, \hat{z}_{q_t}, j)$ computes $\hat{z}$ on a $(1 - N^{-\gamma})$ fraction of inputs from $[N]$. Note that by our choice of γ and ν and the facts that $N \geq n^{27}$ and $m \leq h \leq m^3$, it holds that $N^\nu \geq m^3$ and $N^\gamma \geq d^2$, as desired by the claim.

Let $r_2 = r_{\mathsf{pre}} + r_{\mathsf{main}}$. S_2 takes $(\alpha_{\mathsf{pre}}, \alpha_{\mathsf{main}}) \in \{0,1\}^{r_2}$ as input, it first runs $S_{\mathsf{pre}}(\alpha_{\mathsf{pre}})$ to compute $q_1, q_2, \ldots, q_t \in [N]$, and then runs $S_{\mathsf{main}}(\alpha_{\mathsf{main}})$ to obtain the oracle circuit C_2', then it constructs the desired circuit E_2 that first computes $\hat{z}_{q_1}, \ldots, \hat{z}_{q_t}$, and then outputs C_2 by fixing the first t bits of the input to C_2' to $\hat{z}_{q_1}, \ldots, \hat{z}_{q_t}$. Note that C_2 is a m^μ-size $\mathcal{TC}^0_{d_0}$ circuit.

By Fact B.2, E_2 is a $\mathrm{poly}(m)$-size $\mathcal{TC}^0_{O(d_0)} \circ \mathsf{XOR}$ circuit, S_2 can be implemented by an $n \cdot \mathrm{poly}(m)$-size $\mathcal{TC}^0_{O(d_0)}$ circuit. ■

Let r_3 be the number of random bits used by $\mathsf{RM\text{-}LDC}_{p,\ell,d}$. Finally, S takes $(\alpha_1, \alpha_2, \alpha_3) \in \{0,1\}^{r_1} \times \{0,1\}^{r_2} \times \{0,1\}^{r_3}$ as input, and computes $E_1 = S_1(r_1)$, $E_2 = S_2(r_2)$, and C_3 by fixing the randomness in $\mathsf{RM\text{-}LDC}_{p,\ell,d}$ (Lemma 5.4) by α_3. It then constructs the final circuit E on input z that operates as follows: compute $C_1 = E_1(z)$, $C_2 = E_2(z)$, and compute an oracle circuit

$$C'^O(\vec{u}) := C_3^{C_2^{C_1^O}}(\vec{u})$$

for $\vec{u} \in \mathbb{F}^m$, where $O\colon \{0,1\}^m \to \{0,1\}$ is an oracle. Output

$$C^O(i) = (C')^O(\xi(i)).$$

The complexity and correctness of S follows from the two claims above, and from Lemma 5.4. ■

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il