Electronic Colloquium on Computational Complexity, Report No. 208 (2023)

Opening Up the Distinguisher: A Hardness to Randomness Approach for $\mathsf{BPL} = \mathsf{L}$ that Uses Properties of BPL

Dean Doron*
Ben Gurion University
deand@bgu.ac.il

Edward Pyne†
MIT
epyne@mit.edu

Roei Tell
University of Toronto
roei@cs.toronto.edu

Abstract

We provide compelling evidence for the potential of hardness-vs.-randomness approaches to make progress on the long-standing problem of derandomizing space-bounded computation.

Our first contribution is a derandomization of bounded-space machines from hardness assumptions for classes of *uniform deterministic algorithms*, for which strong (but non-matching) lower bounds can be unconditionally proved. We prove one such result for showing that $\mathsf{BPL} = \mathsf{L}$ "on average", and another similar result for showing that $\mathsf{BPSPACE}[O(n)] = \mathsf{DSPACE}[O(n)]$.

Next, we significantly improve the main results of prior works on hardness-vs.-randomness for logspace. As one of our results, we relax the assumptions needed for derandomization with minimal memory footprint (i.e., showing $\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}[c \cdot S]$ for a small constant c), by completely eliminating a cryptographic assumption that was needed in prior work.

A key contribution underlying all of our results is *non-black-box use of the descriptions of space-bounded Turing machines*, when proving hardness-to-randomness results. That is, the crucial point allowing us to prove our results is that we use properties that are specific to space-bounded machines.

*Supported in part by NSF-BSF grant #2022644.
†Supported by a Jane Street Graduate Research Fellowship.

ISSN 1433-8092

Contents

1 Introduction

Determining the power of randomness in space-bounded computation has been a long-standing challenge in complexity theory, and indeed, over the past several decades, there has been a substantial amount of work attempting to prove $\mathbf{RL} = \mathbf{L}$ and $\mathbf{BPL} = \mathbf{L}$. This work has resulted in a wide range of unconditional partial results, via pseudorandom generators [Nis92; INW94; NZ96; FK18; CHH+19; CLT+23], other pseudorandom objects [BCG20; HZ20; CHL+23; CL23], non-black-box derandomizations [Nis94; SZ99; Hoz21; CDS+23; PP23], and derandomization of specific problems in $\mathbf{BPL}$ [Rei08; AKM+20].

However, in sharp contrast to the time-bounded case, there has been relatively little work so far exploring the "hardness to randomness" paradigm in the context of space bounded derandomization. Two decades ago, Klivans and van Melkebeek [KM02] showed that $\mathbf{BPL} = \mathbf{L}$ follows from exponential circuit lower bounds for linear space, but until recently, there have been few other works exploring this path.

This is, perhaps, understandable. In light of the unconditional progress on $\mathbf{BPL}$ vs. $\mathbf{L}$, it was unclear if the hardness to randomness paradigm is necessary (in particular, since the hardness assumptions seemed out of reach). Another reason is that the PRG constructions (and non-black-box derandomizations) for $\mathbf{BPL}$ exploited the fact that $\mathbf{BPL}$ algorithms use their randomness in a *read-once* fashion, but it was previously unknown how to exploit this for hardness to randomness results (see, e.g., [HH23, Section 4.3]).

Two recent works revisited the study of hardness to randomness for space-bounded computation, driven by new motivations. Doron and Tell [DT23] did so for the purpose of obtaining stronger derandomization, namely one with minimal memory footprint (following analogous work in the time-bounded setting [DMO+22; CT21b; CT21a]). Pyne, Raz, and Zhan [PRZ23] did so in order to construct algorithms that *certify* the correctness of derandomization: Their algorithm either derandomizes successfully, or explicitly refutes a hardness assumption. Indeed, joining these new motivations, the obvious and long-standing motivation for studying such approaches is the following question: Can we find new ways for making progress on $\mathbf{BPL} = \mathbf{L}$?

Our contributions: A bird's eye view. In this work we provide compelling evidence that the hardness-to-randomness paradigm can drive progress on unconditional derandomization of space-bounded computation. In particular:

1. We prove that derandomization of $\mathbf{BPL}$ follows from remarkably weak lower bounds: In particular, from lower bounds for deterministic uniform models, for which strong (but non-matching) lower bounds can already be unconditionally proved.

2. We significantly improve the main results of prior works on hardness-vs.-randomness for logspace [DT23; PRZ23], deriving minimal-memory derandomization, and certified derandomization, from fewer and weaker assumptions.

To obtain these results, our proofs indeed exploit perhaps the most important weakness of the **BPL** model – the read-once nature of the distinguisher.

1.1 Derandomization from Hardness for Deterministic Uniform Algorithms

Our first main result deduces derandomization of **BPL** from lower bounds for a class of *deterministic and nearly-uniform* constant-depth circuits that are aided by a bounded-space oracle. To define this model, let us first define a class of logspace uniform TC^0 circuits, where we permit a logarithmic amount of nonuniform advice.

Definition 1.1 (logspace-uniform bounded-space machines)**.** *We say that* $\{C_n\}_{n\in\mathbb{N}}$ *is a family of* log-spaceadvice-uniform TC^0 circuits *of size* $T = T(n)$ *and depth* $d = O(1)$ *if there is some constant* $C > 1$ *and a Turing machine* M *such that the following holds. On input* 1^n*,* M *gets* $C \cdot \log T$ *bits of non-uniform advice, runs in space* $C \cdot \log T$*, and prints the* TC^0 *circuit* C_n*.*

The required lower bounds will be for log-spaceadvice-uniform TC^0 circuits of fixed polynomial size (say, n^c) that can make oracle queries to a function computable in fixed logspace (say, $c \cdot \log n$).[1] This model is quite weak, and in particular, we can unconditionally prove strong lower bounds for it. In fact, following the approach of Santhanam and Williams [SW13], we show lower bounds for this model in a class as weak as **L**:

Proposition 1.2 (see Section 6.3)**.** *For every* $c, c', d \in \mathbb{N}$*, there is a language in* **L** *that cannot be solved by log-spaceadvice-uniform* $(\mathsf{TC}^0)^{\mathsf{DSPACE}[c'\cdot\log n]}$ *circuits of size* n^c *and depth* d*.*

Our main result is that improving the uniformity condition from **L** to logspace-uniform TC^0, and strengthening the bound from worst-case hardness to mild average-case, *suffices to derandomize* **BPL** on average, with success probability arbitrarily close to 1:

Theorem 1. *Assume that for every constant* $c \in \mathbb{N}$ *there exist constants* $k, d \in \mathbb{N}$ *and* $\delta > 0$*, and a function* $f : \{0,1\}^n \to \{0,1\}$ *that satisfies the following.*

1. ***Upper bound.*** f *is computable in logspace-uniform* TC^0 *of depth* d *and size* $O(n^k)$*.*
2. ***Lower bound.*** *For every log-spaceadvice-uniform* $(\mathsf{TC}^0)^{\mathsf{DSPACE}[c\cdot\log(n)]}$ *circuit family* $\{C_n\}$ *of size* n^c *and depth* $c \cdot d \cdot k^2$*, and every sufficiently large* $n \in \mathbb{N}$*, we have*

$$\Pr_{x \leftarrow U_n}[C_n(x) \neq f(x)] \geq \delta.$$

Then, $\mathsf{BPL} \subseteq \bigcap_{\varepsilon>0} \mathsf{avg}_\varepsilon \mathsf{L}$.

[1] A circuit family in this model is defined by a Turing machine (and a sequence of short advice strings) that prints the circuit, and another Turing machine that answers the oracle queries. Alternatively, we can assume that the first machine prints a description of the second machine along with the circuit.

The class $\mathsf{avg}_\varepsilon\mathbf{L}$ consists of all languages that can be decided by a deterministic logspace algorithm correctly on at least $1-\varepsilon$ of the inputs, and with zero error (i.e., on every input the algorithm either decides the language correctly or outputs $\perp$; see Definition 3.8).

The assumption in Theorem 1 deviates sharply from previous hardness vs. randomness results, because it only requires lower bounds for *deterministic* algorithms with a *logarithmic* amount of advice. Known results, both in the time-bounded setting and in the space-bounded setting, require lower bounds either for fully non-uniform models (e.g., hardness in $\mathbf{E}$ for exponential-sized circuits, as in [NW94; IW97; SU05; Uma03], and in [KM02; DT23]) or for probabilistic models (e.g., hardness in high deterministic time for lower probabilistic time, as in [IW98; TV07; CRT+20], and in [CT21a; LP22a; LP22b; CRT22]). We are able to rely on this inherently weaker assumption by exploiting the fact that we are derandomizing a $\mathbf{BPL}$ algorithm; in fact, we do so in a non-black-box way with respect to the $\mathbf{BPL}$-machine (see Section 2.1 for details).

As far as we are aware, our work is the first to study average-case derandomization of $\mathbf{BPL}$. We view this as an advantage, as it offers a potentially tractable avenue for progress. Also, as mentioned above, Theorem 1 obtains derandomization with zero error. It turns out that this feature does not indicate an excessively strong assumption: We show a tight connection between average-case derandomization and average-case derandomization with zero-error when derandomizing $\mathbf{BPL}$ (for details, see Section 6.5).

Theorem 1 answers an open problem posed Chen and Tell [CT21a], who asked whether a uniform hardness-vs.-randomness approach could be applied to the $\mathbf{BPL} = \mathbf{L}$ question. (In fact, they asked whether it could be applied with lower bounds for uniform *probabilistic* algorithms, whereas we prove that lower bounds for deterministic algorithm suffice.)

The scaled-up setting: Derandomizing linear space. Our second main theorem deduces *worst-case* derandomization of *linear space* (rather than logspace) from a hardness assumption that, yet again, seems remarkably close to what can be actually proved.

The assumption refers to worst-case hardness for deterministic and fully uniform algorithms. To state the result, let us say that a $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuit family $\{C_n\}$ is logspace-uniform and of size $S = S(n)$ if there is a machine that gets input 1^n, runs in space $O(\log S)$, and prints both the $\mathbf{TC}^0$ circuit and the ROBP, and its total output length, i.e., the description lengths of the circuit and the ROBP, is S. Then:

Theorem 2. *Assume that there are $\varepsilon > 0$ and $L \in \mathbf{DSPACE}[O(n)]$ such that L is hard for logspace-uniform $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuits of size $2^{\varepsilon \cdot n}$ and depth d (for some universal constant $d \in \mathbb{N}$), on all but finitely many input lengths. Then, $\mathbf{BPSPACE}[O(n)] = \mathbf{DSPACE}[O(n)]$.*

We find Theorem 2 striking: It means that *the only thing* standing between a trivial diagonalization argument and proving $\mathbf{BPSPACE}[O(n)] = \mathbf{DSPACE}[O(n)]$ is the fact that the circuit in the lower bound is printed by an algorithm using space $C \cdot n$, whereas the upper bound uses space $c \cdot n$, where $c < C$.[2] Proving lower bounds when this is the only

[2] Recall that $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuits of size $2^{\varepsilon \cdot n}$ and depth d can be simulated in space $O(\varepsilon \cdot d \cdot n)$, using

gap is not difficult in the setting of polynomial time (rather than exponential time), as has been shown by Santhanam and Williams [SW13] (see, e.g., Section 6.3).

Another reason for hope is that, similarly to average-case derandomization of **BPL**, derandomization of *linear space* is a relaxed goal that, as far as we are aware, has not been extensively studied. (Recall that **BPL** = **L** implies **BPSPACE**$[O(n)]$ = **DSPACE**$[O(n)]$.)

Certified derandomization with stronger guarantees. As an immediate application of some of the new technical components in the proofs of Theorems 1 and 2, we improve the main result of [PRZ23]. They showed that for every language L in **BPL**, there is a deterministic logspace algorithm D that, on input x of length n, either decides L on x, or prints a circuit of size $2^{\varepsilon \cdot \ell}$, where $\varepsilon > 0$ may be an arbitrarily small constant, for a linear-space-complete language L_{hard} on inputs of size $\ell = \Theta(\log n)$. We directly improve this tradeoff, so that D either decides L on x or prints a $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuit of size $2^{\varepsilon \cdot \ell}$ (rather than a general circuit; see Section 2.1.2 and Theorem 5.1 for details).

1.2 Minimal-Memory Derandomization From Weaker Assumptions

Our second set of results focuses on derandomization with minimal memory-overhead, as introduced by Doron and Tell [DT23] (following a line of work studying "superfast" derandomization in the time-bounded setting [DMO+22; CT21b; CT21a; CT23b]). Our main contribution is in showing that derandomization with minimal memory overhead can be obtained from considerably weaker assumptions.

Recall that minimal memory-overhead derandomization (or, "derandomization with minimal memory footprint") has the goal of showing **BPSPACE**$[S] \subseteq$ **DSPACE**$[c \cdot S]$ for a small constant c. Doron and Tell [DT23] showed that $c \approx 2$ is possible, given two assumptions. The first is high-end nonuniform hardness of a language in linear space:

Assumption 1. *For a sufficiently large constant C there exists a language L computable in deterministic space $(C+1) \cdot n$ that is hard for algorithms that run in deterministic space $C \cdot n$ with $O(2^{n/2})$ bits of advice.*

Their second assumption asserts that there exists a *strongly explicit cryptographic PRG*, with arbitrary polynomial stretch $\{0,1\}^{n^\eta} \to \{0,1\}^n$; specifically, the PRG that they need has to be computable in space $O(\eta \cdot \log(n)) + O(\operatorname{loglog} n)$. (One candidate that they suggested is Goldreich's expander-based PRG [Gol11b].)

From the combination of both hypotheses, they deduce derandomization with a multiplicative overhead in space of $(2 + c/C)$, for a universal constant c. Moreover, if the hard function was computable in catalytic space in a particular setting of parameters, they obtained an overhead of $(1 + c/C)$ (for more on the catalytic model, see Section 1.3).

the standard DFS-style bounded-space simulation of low-depth circuits. Thus, the only obstruction for diagonalization is the complexity of the logspace machine printing the circuit.

Removing the cryptographic assumption: Derandomization from non-uniform hardness alone. Our first main result in this context *completely eliminates* the cryptographic assumption, and deduces high-end derandomization solely from Assumption 1:

Theorem 3. *Suppose that Assumption 1 is true with some constant $C > 1$. Then, for $S(n) = \Omega(\log n)$ and for a universal constant $c > 1$, we have that*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[\left(2 + \frac{c}{C}\right) \cdot S\right].$$

Moreover, if the language L is computable in $\mathsf{CSPACE}[\delta n, (C + \delta + 1)n]$,[3] *then*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[\left(1 + \frac{c}{C}\right) \cdot S\right].$$

As in Section 1.1, the assumption in Theorem 3 deviates sharply from analogous known results. Similarly to [DT23], other prior works concerning derandomization with little computational overhead relied either on a combination of cryptographic assumptions and of worst-case hardness for non-uniform procedures [CT21b; CT21a] or on hardness for *non-deterministic* non-uniform procedures [DMO+22]. This is not a coincidence: For the time-bounded setting of [DMO+22; CT21b; CT21a], Shaltiel and Viola [SV22] proved a barrier, asserting that the relevant conclusion (i.e., "superfast" derandomization) cannot be deduced by an algorithm that is analyzed via the standard hardness-to-randomness approach (i.e., an approach that includes black-box hardness amplification and involves the hybrid argument; see [SV22] for details).

In contrast to prior works, Theorem 3, which refers to the space-bounded setting, only relies on worst-case hardness for standard non-uniform space-bounded procedures. Indeed, a crucial part of our derandomization algorithm is *not* analyzed via the standard hardness-vs.-randomness approach (and, in particular, does not involve the standard hybrid argument). Our proof relies on the fact that we are derandomizing a space-bounded algorithm, and we rely on recent works studying pseudorandomness for generalizations of the read-once branching program model [FK18; CLT+23].

Alternative assumptions: Minimal-memory derandomization from hardness of compression. In addition to the result mentioned above, [DT23] also derive minimal memory derandomization from the combination of a cryptographic PRG, and a uniform assumption regarding the hardness of compression of a multi-output function by randomized algorithms. We obtain the same result from hardness of compression by *deterministic* algorithms, plus a hitting-set generator (HSG) that suffices to prove the "standard" version of $\mathsf{BPL} = \mathsf{L}$:

Assumption 2. *There is a $(1/n)$-HSG $H\colon \{0,1\}^{O(\log n)} \to \{0,1\}^n$ for NC^2 circuits of size n. Moreover, H can be computed in space $O(\log n)$.*

[3] The notation CSPACE refers to catalytic space; see Section 1.3, and Section 3.5 for the definition.

An HSG as the one in Assumption 2 is implied, for example, by the assumptions originally used in [KM02] to deduce that $\mathsf{BPL} = \mathsf{L}$. However, while this HSG suffices to prove that $\mathsf{RL} = \mathsf{L}$ (and $\mathsf{BPL} = \mathsf{L}$, by [CH22]), it is entirely unclear how to use it to prove derandomization of BPL with minimal memory footprint.

The assumption that will allow us to deduce derandomization with minimal memory footprint, rather than just $\mathsf{BPL} = \mathsf{L}$, refers to hardness of deterministic compression:

Assumption 3. *For all constants $c > 0$, $\varepsilon \in (0,1)$, and $C \in \mathbb{N}$, the following holds. There exists a function $f\colon \{0,1\}^\star \to \{0,1\}^\star$, such that f maps n bits to n^2 bits, and is computable in*

$$\mathsf{DSPACE}\left[\frac{C+1+\varepsilon+\delta}{2} \cdot \log n\right]$$

for some constant $\delta > 0$. Moreover, for every deterministic algorithm that runs in space $c \cdot C \log n$, there are at most finitely many $x \in \{0,1\}^\star$ for which $R(x)$ prints a Turing machine M of description size $O(|x|)$ that runs in space $\frac{C+1+\varepsilon}{2} \cdot \log |x|$, such that M prints $f(x)$.

Compared to Assumption 3 of [DT23], we require the compression algorithm to be *deterministic*, rather than randomized. However, note that we allow the compression algorithm to use an arbitrary constant factor more space.[4] Then, our result is:

Theorem 4. *Suppose that Assumption 2 and Assumption 3 are true. Then, for $S(n) = \Omega(\log n)$ we have that*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[\left(2 + \frac{c}{C}\right) S\right]$$

where $c > 1$ is a universal constant. Moreover, if f is computable in $\mathsf{CSPACE}[\delta n, (C+\delta+1)n]$, then $\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[\left(1 + \frac{c}{C}\right) \cdot S\right]$.

1.3 A New Proof of $\mathsf{BPL} \subseteq \mathsf{CL}$

Finally, we explore what other consequences we can obtain from our derandomization primitives. In the catalytic computing model of Buhrman *et al.* [BCK+14], we are given $O(\log n)$ bits of standard workspace, and a *catalytic* tape $\mathbf{w}$ of length n^c, which functions as follows. The tape $\mathbf{w}$ is initialized to an arbitrary value, and we may edit it during the computation, but must exactly reset the tape to the original configuration at the end.

The work of [BCK+14] proved that logspace-uniform $\mathsf{TC}^1 \subseteq \mathsf{CL}$ (and thus $\mathsf{BPL} \subseteq \mathsf{CL}$[5]). Another known proof of $\mathsf{BPL} \subseteq \mathsf{CL}$ relies on treating the catalytic tape as a collection of random walks (see the survey by Mertz [Mer23] for a sketch). We give a new proof, which uses only two features of the BPL model: For a randomized logspace machine M, we can evaluate $M(x,r)$ in L given x and r; and we have a deterministic *distinguish-to-predict transformation* for read-once branching programs computable in L.

[4] The additional space of the compression algorithm, captured by the constant c, is proportional to the constant hidden in the $O()$ notation of Assumption 2. Thus, the quality of the HSG from Assumption 2 determines the space overhead of the compression algorithm from Assumption 3.

[5] As approximate iterated matrix squaring can be performed in TC^1.

Definition 1.3 (informal; see Definition 8.2)**.** *A (black-box) distinguish-to-predict transformation for $\mathbf{C}$ circuits is a deterministic algorithm that, given $C \in \mathbf{C}$ of size n, outputs a collection $P_1, \ldots, P_{\mathrm{poly}(n)}$ of $\mathbf{C}$ circuits of size $\mathrm{poly}(n)$ such that for any distribution D, if D does not fool C, there is some i such that P_i is a previous-bit-predictor for D.*

One of our main technical tools can be stated simply in this language:

Theorem 1.4 (informal, see Theorem 4.2)**.** *There is a logspace-computable distinguish-to-predict transformation for read-once branching programs.*

We then show that the existence of such a transformation suffices for derandomization:

Theorem 1.5 (see Section 8)**.** *Suppose a class of circuits $\mathbf{C}$ satisfies the following.*

1. *There is a* **CL** *algorithm that, given $C \in \mathbf{C}$ and $r \in \{0,1\}^n$, outputs $C(r)$.*

2. *There is a* **CL***-computable distinguish-to-predict transformation for $\mathbf{C}$ circuits.*

Then, there is a **CL** *algorithm that, given $C \in \mathbf{C}$, outputs $\mathbb{E}_r[C(r)]$ up to error $1/6$.*

Corollary 1.6. *It holds that* **BPL** $\subseteq$ **CL***.*

Our proof strategy provides a possible line of attack on a natural question related to catalytic logspace (see, e.g., [Mer23, Problem 16]). While logspace-uniform $\mathbf{NC}^1$ is contained in **L**, it is not known that logspace-uniform *randomized* $\mathbf{NC}^1$ (wherein the circuits take random bits as auxiliary input) is contained in **BPL**, or even in **CL**. This is because $\mathbf{RNC}^1$ circuits can read their random bits multiple times, whereas **BPL** machines cannot.

If a **CL**-computable distinguish-to-predict transformation existed for any class of circuits for which we can evaluate C-circuits in **CL** (for instance, $\mathbf{NC}^1$), we would obtain that the randomized analogue of that class lies in **CL**. Note that we can tolerate both circuit evaluation and the distinguish-to-predict transformation being computable in **CL**, whereas for ROBPs we have that both are computable in **L**.

2 Technical Overview

In Section 2.1 we outline the proof of Theorem 1. Throughout the description we also introduce some of the new technical tools that will be used for the proofs of our other results. The proof ideas of Theorem 3 and Theorem 4 are presented in Section 2.2, and the proof outline of Corollary 1.6 is presented in Section 2.3.

2.1 Derandomization from Hardness for Deterministic Uniform Algorithms

Our derandomization algorithm will be based on reconstructive *targeted pseudorandom generators*. Targeted PRGs were introduced by Goldreich [Gol11d; Gol11c], and recent works presented various constructions of targeted PRGs that are pseudorandom under hardness assumptions; see, e.g., [CT21a; SM23; CT23b; CLO+23] and see [CT23a] for a survey. A reconstructive targeted generator G is based on a hard function f. The generator gets an input x, and outputs a list of strings, hoping that the list will be pseudorandom[6] for every *uniform* algorithm that also has access to the same input x. Correctness is established via a reduction to the hardness of f: If an efficient uniform probabilistic algorithm $A(x, \cdot)$ distinguishes the output-list of $G(x)$ from uniformly random strings, then we can compute f on x by an efficient algorithm $F(x)$. We stress that the connection holds instance-wise, i.e., for every fixed x. That is, if $G(x)$ is not pseudorandom for A with input x (i.e., for $A(x, \cdot)$), then $F(x)$ computes $f(x)$ too efficiently, contradicting the assumed hardness of f on x.

The reduction from distinguishing $G(x)$ to computing $f(x)$ is called a *reconstruction procedure*. Known reconstruction procedures are either non-uniform or probabilistic, necessitating hardness assumptions for non-uniform circuits or for probabilistic algorithms (see, e.g., [CT21a; LP22a; LP22b; DT23; SM23; CLO+23; CT23a; CTW23]).[7]

Consistency tests and deterministic reconstruction. The starting point for our results is a recent work of Pyne, Raz, and Zhan [PRZ23]. They showed that when A is a small-space algorithm, the reconstruction procedure of the classical Nisan–Wigderson PRG [NW94] can be made *almost-deterministic*; that is, the reconstruction procedure is a small-space algorithm, and the number of random coins that it uses does not exceed its space complexity, and thus it is possible to enumerate over all random choices.

Their result crucially relies on the fact that A is a small-space algorithm, implying that $D_x(\cdot) = A(x, \cdot)$ is computable by a bounded-width *read-once branching program* (ROBP). The crux of their proof is a simple combinatorial lemma, showing that deciding whether a distribution $\mathbf{w}$ is pseudorandom for $D_x(\cdot)$ reduces to a small number of "consistency tests" that can be performed using the description of D_x. Such tests have been explored before, and have found multiple applications [Nis93; CH22; GRZ23; PRZ23].

A motivating observation for our work is that the result of [PRZ23] can be used to deduce derandomization from lower bounds for logspace-uniform circuits. Specifically, we can deduce the following as a corollary: If $\mathbf{DSPACE}[O(n)]$ is hard for *logspace-uniform* circuits of size $2^{\varepsilon \cdot n}$ for some $\varepsilon > 0$, then $\mathbf{BPSPACE}[O(n)] = \mathbf{DSPACE}[O(n)]$.[8]

[6] By "pseudorandom list" we mean that the uniform distribution over the list aims to fools the algorithm. Stated differently, G gets x and a random seed s, and the pseudorandom distribution is generated by choosing the seed s uniformly at random and outputting $G(x, s)$.

[7] The situation is similar when considering reconstruction procedures for classical (i.e., non-targeted) pseudorandom generators; see, e.g., [NW94; IW97; IW98; KM02; STV01; TV07; SU05; Uma03].

[8] The approach for doing so is demonstrated in the proof of Theorem 6.16 (which asserts a significantly

This latter statement is, of course, still far from what we want: The hardness is uniform circuits of exponential size, and for general circuits rather than bounded space, or small-depth, models. Fortunately, recent works in derandomization suggest a way to avoid these shortcomings: When using *targeted* PRGs, it may suffice to rely on hardness assumptions for uniform circuits of polynomial size (see, e.g., [CT21a; LP22a; LP22b; SM23]), and moreover, there are already known targeted PRGs whose reconstruction procedure yields constant-depth circuits, rather than general circuits [CTW23].

The pressing question is whether we can materialize this approach in the setting of derandomizing small space: Can we design a *targeted* PRG that has an *almost-deterministic* reconstruction yielding constant-depth circuits? And would this be enough to derandomize **BPL** from hardness against weak, uniform, and deterministic algorithms?

Our technical contribution. The main technical contribution underlying Theorem 1 is a new variant of the Chen–Tell targeted hitting-set generator (HSG) [CT21a]. Our new variant has an *almost-deterministic* reconstruction procedure yielding a *constant-depth circuit*, when the intended pseudorandomness is for bounded-space machines. We prove:

Theorem 2.1 (an almost-deterministic targeted somewhere-PRG; see Theorem 6.7)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}^{r(n)}$ be computable by logspace-uniform TC^0 circuits of size $T(n) = \mathrm{poly}(n)$ and depth $d = O(1)$, and let $m = T^{\delta}$ for a sufficiently small constant $\delta > 0$. Then, there are deterministic algorithms G_f, R_f, and $\mathcal{O}_f$ such that:*

1. ***Generator.*** *On input $x \in \{0,1\}^n$, the algorithm G_f runs in space $O(\log T)$ and prints $O(d/\delta^2)$ lists of m-bit strings.*
2. ***Reconstruction.*** *The algorithm R_f gets a description of a space-$\log m$ machine M, and a random seed $y \in \{0,1\}^{O(\log T)}$. It runs in space $O(\log T)$ and prints an oracle TC^0 circuit C_y of size $\mathrm{poly}(n \cdot m) \ll T$ such that for every fixed input $x \in \{0,1\}^n$ the following holds.*

   If $M(x,\cdot)$ is a $\frac{1}{10}$-distinguisher for each of the output lists of $G_f(x)$[9], then with probability at least 0.99 over y it holds that $C_y^{\mathcal{O}}(x) = f(x)$.
3. ***Oracle.*** *The machine $\mathcal{O}_f$ runs in space $O(\delta \cdot \log T)$.*

Note that R_f is indeed almost-deterministic, since it uses $O(\log T)$ random coins and $O(\log T)$ space. However, while it has enough space to enumerate over all choices of random coins, it is not a-priori clear how enumeration can be helpful: Recall that R_f does not get any particular input x; on a seed y, it prints a circuit C_y, hoping that $C_y(x) = f(x)$ for all inputs x (or, at least, for many inputs x). More generally, it is not immediately clear how to deduce Theorem 1 from Theorem 2.1, and we will explain this in Section 2.1.3.

stronger result).

[9]For the standard definition of distinguishers, see Definition 3.1. When we say that $M(x,\cdot)$ is a distinguisher for each of the lists, we mean that for each $i \in [O(d/\delta^2)]$, $M(x,\cdot)$ distinguishes the uniform distribution on the i^{th} output list $G_f(x)_i$ from a uniformly random string.

We stress that the reconstruction R_f is *inherently non-black-box* with respect to the distinguisher M: It crucially uses the description of M, and moreover, we do not know how to prove a similar result when M is not a small-space machine. This is the first construction of a targeted generator in which the reconstruction uses the distinguisher in a non-black-box way, and relies on the distinguisher being in a restricted class.[10]

In Section 2.1.1 we give a high-level overview of the generator's construction, and then in Section 2.1.2 we describe the new technical contributions, and in Section 2.1.3 we explain how to deduce Theorem 1. Jumping ahead, key technical challenges that will be handled include:

- A required improvement to the previously known consistency tests for ROBPs. We present an alternative test that is considerably simpler and more efficient.
- Extending the construction of the targeted HSG of [CT21a] so that it works even when the hard function is in logspace-uniform TC^0.[11]
- Constructing an error-correcting code that has a local list-decoding algorithm computable by uniform, almost-deterministic constant depth circuits. Our code builds on, and improves upon, the constructions of Doron and Tell [DT23] and of [CTW23].
- Showing how to use the targeted HSG for Theorem 1, and in particular how to derandomize **BPL** rather than only **RL**. Previous works relying on [CT21a] deduced derandomization of algorithms with one-sided error, or used advice in the derandomization [CT21a; CRT22; CT23b; SM23; CLO+23].

2.1.1 A quick overview of the generator

The construction follows the general outline of [CT21a], which is inspired by the doubly efficient proof system of Goldwasser, Kalai, and Rothblum [GKR15]. Let C_n be a circuit of size T and depth d computing the hard function f, fix an input $x \in \{0,1\}^n$, and consider the values of all gates in $C_n(x)$ (i.e., in the computation of C_n on x).

A *bootstrapping system* for $C_n(x)$ is a sequence $w_x^{(1)}, \ldots, w_x^{(\bar{d})}$ of strings of length $\mathrm{poly}(T)$, which we also think of as functions $\{0,1\}^{O(\log T)} \to \{0,1\}$, satisfying the following:

1. **Base case.** The function $w_x^{(1)}$ is easily computable, given access to x.
2. **Downward self-reducibility.** There is an efficient procedure that computes $w_x^{(i)}$ given oracle access to $w_x^{(i-1)}$.

[10]The only other work that used a reconstructive targeted generator where the reconstruction uses the distinguisher in a non-black-box way is that of Liu and Pass [LP22a]. In their work, the distinguisher may be an arbitrary Turing machine.

[11]This was not known before. The closest related variation of the [CT21a] targeted HSG was by Chen, Tell, and Williams [CTW23]: Their construction only works with TC^0 functions meeting a stricter uniformity condition, but its (probabilistic) reconstruction procedure also meets a stricter uniformity condition.

3. **Faithful representation.** There is an efficient procedure that computes $C_n(x)$ given oracle access to the last string $w_x^{(\bar{d})}$.

4. **Layer reconstruction.** There is an efficient procedure that gets oracle access to a string $\widetilde{w}_x^{(i)}$ that agrees with $w_x^{(i)}$ on $1/2 + T^{-0.01}$ fraction of its coordinates, and oracle access to $w_x^{(i)}$, and outputs a *small* circuit C_i such that $C_i^{\widetilde{w}_x^{(i)}}$ computes $w_x^{(i)}$ correctly on all coordinates.

Above, whenever we say "an efficient procedure", one can think of a uniform TC^0 circuit of size T^{δ} for a small $\delta > 0$. Note that any C_n that has an efficient bootstrapping is efficiently computable, simply by following the sequence of $\bar{d}$ reductions, but such a naive implementation is too costly for us.

Going back to Theorem 2.1, the generator G_f uses each of the $\bar{d} = O(d)$ functions as a hard function for the NW generator, and outputs the union of the $\bar{d}$ output lists generated by the NW PRG. The reconstruction procedure, reducing pseudorandomness $G(x)$ to hardness of $C_n(x)$, works as follows. Assume that a distinguisher D distinguishes all $\bar{d}$ output lists from uniform, and let us compute C_n on input x. Iteratively, for $i = 1, \ldots, \bar{d}$, we construct a small circuit C_i such that C_i^D computes the i^{th} function $w_x^{(i)}$ in the sequence:

1. The base case follows trivially (since the first function is easily computable from x).

2. Given the pre-constructed circuit C_{i-1} and a distinguisher D for NW with $w_x^{(i)}$, we can quickly build a circuit C_i such that C_i^D computes $w_x^{(i)}$ (this relies on the classical reconstruction procedure of [NW94], on the layer reconstruction procedure, and on the reduction of computing $w_x^{(i)}$ to computing $w_x^{(i-1)}$).[12]

3. Finally, the last function allows us to compute the output of $C_n(x) = f(x)$.

To illustrate how this contradicts the hardness of f, let us implement the procedure above by a small uniform TC^0 circuit C, under the assumption that all procedures in the bootstrapping system are uniform TC^0 circuits of size T^{δ}. The circuit C works in $\bar{d}$ steps, where in each step $i \in [\bar{d}]$ it computes a description of C_i. Each step can be done in size $T^{O(\delta)}$, by implementing the procedures of the bootstrapping system, the reconstruction of [NW94], and simulating C_{i-1}. Thus, the overall depth of C is $O(\bar{d})$, its size is $T^{O(\delta)} \ll T$, and it uses oracle access to the distinguisher D. If we assume that $C_n(x)$ cannot be computed by such circuits, we get a contradiction, and deduce that at least *one* of the output lists, corresponding to some $w_x^{(i)}$, must be pseudorandom for D.

[12]Specifically, the reconstruction procedure of NW allows to obtain a small circuit C_{NW} such that C_{NW}^D agrees with $w_x^{(i)}$ on $1/2 + T^{-.01}$ of the inputs $j \in [\text{poly}(T)]$. By combining this with the layer reconstruction, we obtain a small circuit C_i such that C_i^D agrees with $w_x^{(i)}$ on all $j \in [\text{poly}(T)]$. Both procedures require oracle access to $w_x^{(i)}$ to work correctly, and we can supply it using the downward self-reducibility procedure, the pre-computed description of C_{i-1}, and our oracle access to D (to compute $C_{i-1}^D(j) = w_x^{(i-1)}(j)$).

2.1.2 New technical tools and implementation ideas

The original work of [CT21a] applies to general circuits, or to NC circuits, and has a probabilistic reconstruction procedure. A recent work of Chen, Tell, and Williams [CTW23] constructs a bootstrapping system for uniform TC^0 circuits, but for circuits that meet a stricter uniformity condition than ours, and such that all relevant procedures in the system are in *probabilistic* uniform TC^0.[13] Their work serves as our technical starting point.

To prove Theorem 2.1 we construct a bootstrapping system for any *logspace-uniform* TC^0 circuit C_n wherein all the relevant procedures can be computed in *almost-deterministic* logspace-uniform TC^0 of size $\mathrm{poly}(n, T^\delta)$, with oracle access to a function $\mathcal{O}$ computable in space $O(\delta \cdot \log T)$. Since our distinguisher $M(x, \cdot)$ is a machine computable in space $O(\delta \cdot \log T)$, the oracle $\mathcal{O}$ does not degrade our assumption.[14]

The construction of the bootstrapping system and of the targeted generator are presented in Section 6.1 (see, in particular, Proposition 6.5 and Theorem 6.7). We now mention some key parts, while providing pointers to the relevant results in Section 6.1.

Canonical form and arithmetization of logspace-uniform TC^0. Loosely speaking, the functions $w_x^{(i)}$ are based on arithmetizing the layers of $C_n(x)$ (i.e., the sequence of gate-values at each layer when computing C_n on x) as low-degree polynomials, and then applying a suitable error-correcting code to them. These ensure that we will have efficient procedures for downward self-reducibility and layer reconstruction, and a major bottleneck in designing them is the fact that the relevant procedures need to compute a suitable low-degree extension of the *circuit-structure function* of C_n (i.e., the function printing the description of C_n).[15]

In our setting, the description of C_n is computable in space $O(\log T)$, but not by TC^0 circuits of size T^δ. To handle this, we exploit the three-components nature of our reconstruction: It consists of an $O(\log T)$-space algorithm R_f printing the circuit C_y, of the TC^0 circuit C_y, and of an oracle $\mathcal{O}$ computable in space $O(\delta \log T)$. We show how to transform every circuit C_n into another circuit C'_n of a "canonical form" such that the circuit-structure function of C'_n has a low-degree extension that can be computed, at any stage of the reconstruction, by one of the three components. Loosely speaking, the description of some gates in C'_n takes space $O(\log T)$ to compute (i.e., it is too costly for C_y and for $\mathcal{O}$), but this description will be queried by C_y in a way that does not depend on the input x,

[13] We note that their reconstruction procedure has to satisfy efficiency properties that ours does not.

[14] One may suspect that an ROBP distinguisher suffices, as in classical PRGs using nonuniform hardness assumptions. However, the circuit that we output does not have any input x hard-wired into it (i.e., it should work correctly for all x, or at least for most x), and neither does the oracle. Thus, we will need an oracle that gets input (x, r) and outputs $M(x, r)$. This issue did not arise in previous works concerning non-black-box derandomization of weak classes (e.g., in [CT21a; CTW23]) since in previous works, the classes did not access their input in a read-once fashion.

[15] This challenge dates back to the original work of [GKR15], who handled it by constructing an additional auxiliary protocol. An alternative solution was suggested by Goldreich [Gol18] (which adds a $\log T$ factor to the depth, and is thus unsuitable for our setting). Alternatively, assuming a sufficiently strict uniformity conditions (as in [CTW23] and in some results of [GKR15]) avoids this problem.

and can thus be computed in advance by R_f; and the description of all other gates in C'_n can be efficiently computed by C_y and $\mathcal{O}$, when they are given a short advice string that can be computed by R_f in advance and hard-wired into C_y. Details appear in Lemma 6.2 and Propositions 6.4 and 6.5.

Consistency tests: Simple and efficient. Recall that each step in the reconstruction procedure will construct a circuit C_i for $w_x^{(i)}$, using C_{i-1} and the distinguisher $M(x, \cdot)$. Each such step relies on the classical reconstruction procedure of [NW94], and thus we need to make the latter procedure almost-deterministic.

The bottleneck in doing so is deterministically transforming any distinguisher for the PRG into a next-bit-predictor. A similar bottleneck was handled in [PRZ23], but their setting was different: In their setting, the reconstruction could evaluate the distinguisher on the output-set of the PRG. In contrast, in our reconstruction:

- R_f does not get any input x, and therefore cannot evaluate a distinguisher $M(x, \cdot)$.
- The small circuit C_y (that does get input x), cannot evaluate the PRG.

To resolve this, we present a new distinguisher-to-predictor transformation, which is simpler and more efficient than previously known ones. Our transformation: (1) Does not need to evaluate the distinguisher on the PRG; (2) Yields a predictor that can be described concisely using only logarithmically many bits; and (3) Better preserves the distinguishing advantage. For now, we rely on Property (1), but we will crucially use Property (2) in Section 2.3. In a gist, our idea is to construct a *previous-bit-predictor* instead of a next-bit-predictor. This turns out to be surprisingly helpful when working with ROBPs. Since the proof is short and self-contained, we refer the reader to Section 4 for further details.

Error-correcting code with derandomized constant-depth decoding. After arithmetization, the second action that will be performed by C_y at each step will be local list-decoding of the error-correcting code. (Recall that each $w_x^{(i)}$ is obtained by applying a code to an arithmetization of a layer of $C_n(x)$.) The code of Chen, Tell, and Williams [CTW23], following Doron and Tell [DT23] and Goldwasser *et al.* [GGH+07], has probabilistic logspace uniform TC^0 decoder, however, we need the decoder to be almost-deterministic.

The key observation is that the code of [CTW23], while complicated, is a combination of many classical codes that are well-understood. Specifically, it uses various combinations of the Reed-Muller code, distance amplification based on expander random walks [ABN+92], the derandomized direct product code of Impagliazzo and Wigderson [IW97], and the Hadamard code. Crucially, inspired by [PRZ23], we use pseudorandomness primitives such as randomness-efficient samplers, and small-biased sets, in order to reduce the number of coins used by the decoder, and manage to construct an almost-deterministic one. We refer the reader to Section 5 for details about the codes constructions, and those will also be used later in Sections 7 and 8.

Separating away the low-success components. The ideas above suffice to obtain an almost-deterministic logspace-uniform $\mathbf{TC}^0$ reconstruction. However, there is a remaining challenge: The reconstruction procedure succeeds only with *low probability*. This is inherent in the uniform reconstruction variant for [NW94], in the local list-decoding of the code, and even the distinguisher-to-predictor transformation yields many candidates (so choosing one at random would yield a low success probability).

The key observation in this context is that we can separate the seed $y \in \{0,1\}^{O(\log T)}$ given to R_f into a long part y_1 of length $O(\log T)$ and a short part y_2 of length $O(\delta \cdot \log T)$ such that the following holds: For every x, with high probability over y_1 there exists y_2 such that $C^{\mathcal{O}}_{y_1,y_2}(x) = f(x)$. Moreover, as has been observed in previous works (e.g., in [CRT22]), roughly speaking, at each step i, the reconstruction procedure in C_y is able to "weed out" a list of candidate circuits for C_i and find a successful one. Since the number of candidate circuits is only $2^{|y_2|} \le T^{O(\delta)}$, we can delegate to C_y the task of enumerating over y_2-s (in parallel) and weeding out the candidate circuits to find a suitable one. Various implementations of this idea appear in Proposition 6.5, Theorem 6.6, and Theorem 6.7.

Where did we use the description of M? The distinguisher-to-predictor transformation from Section 4, which we apply in each step of the reconstruction, crucially depends on having a description of M. Specifically, given input x, the transformation constructs the ROBP defined by $D_x(r) = M(x,r)$, and uses simple manipulations on the nodes of D_x to create a previous-bit-predictor. Our reconstruction R_f thus gets a description of M and hard-wires it into C_y. Whenever C_y needs to compute a previous-bit-predictor on input r, it calls its small-space oracle with a description of M, with the input x, with r, and with the description of the needed manipulations on the nodes of D_x. See the proof of Theorem 6.7 for details.

2.1.3 From targeted generator to derandomization

How do we use Theorem 2.1 to prove Theorem 1? As a first step, consider the derandomization of $\mathbf{RL}$: We simulate an $\mathbf{RL}$ machine M on input x by outputting $\mathsf{OR}_{s \in G_f(x)} M(x,s)$.

Assume towards a contradiction that with high probability over choice of the choice of input $x \in \{0,1\}^n$ we have $\Pr_r[M(x,r)=1] \ge 1/2$, but $M(x,s) = 0$ for all $s \in G_f(x)$. By an averaging argument, there exists a fixed $y \in \{0,1\}^{O(\log T)}$ such that $C_y = R_f(\langle M \rangle, y)$ correctly computes f on most inputs x. By giving this fixed y as advice to R_f, we obtain a log-spaceadvice-uniform circuit family that computes f correctly on most inputs.

This would have been a contradiction, had we assumed that f is hard to compute on most inputs. However, we only assume that f is hard to compute on a small fraction $\delta > 0$ of the inputs. We bridge this gap using the uniform direct-product-based hardness amplification of Impagliazzo *et al.* [IJK+10]. Instead of instantiating the targeted generator with f, we instantiate the generator with the k-wise direct product $f^{\times k}$ of f for an appropriate $k = k(\delta)$, relying on $f^{\times k}$ also being computable in logspace-uniform $\mathbf{TC}^0$. In [IJK+10] they show that if $f^{\times k}$ can be computed on even a small fraction of the inputs, then f can be computed on $1-\delta$ of the inputs with a small number of advice bits. Since our reconstruction uses $O(\log T)$ advice bits anyway, this does not degrade our assumption. See Theorem 6.10 for details.

Derandomizing BPL. The last step is showing how to derandomize **BPL**, rather than only **RL**. Derandomizing algorithms with two-sided error has been a consistent challenge so far in works that used the targeted generator of [CT21a] (see [CRT22; CT23b]).

We are able to overcome this challenge because our reconstruction is almost deterministic. Specifically, instead of outputting $\mathsf{OR}_{s\in G_f(x)} M(x,s)$, the derandomization on input x can iterate over all choices for a seed y, run $R_f(\langle M\rangle, y)$, and check whether or not $C_y(x) = f(x)$. (Indeed, in contrast to R_f, the derandomization algorithm gets an input x, and can thus check if the reconstruction worked on x. This crucially relies on the fact that R_f is almost-deterministic, allowing the derandomization to enumerate over y-s.)

It is still not clear why that would be helpful. The last observation (already mentioned above, following [CRT22]) is that the reconstruction can "self-check": Loosely speaking, given input x and seed y, for each of the $O(d/\delta^2)$ lists that $G_f(x)$ outputs, we can check in space $O(\log T)$ whether or not a corresponding part of $C_y(x)$ is "correct" with respect to computing $f(x)$. The derandomization thus finds a list such that the corresponding part of $C_y(x)$ is incorrect, and this list will be pseudorandom for $M(x,\cdot)$.

Indeed, the above description is informal, and hides some details. The full details appear in the "furthermore" part of Theorem 6.7 and in the proof of Theorem 6.8.

The scaled-up version. The proof of Theorem 2 is different, and significantly simpler. First, it uses a PRG, rather than a targeted PRG (specifically the NW PRG with the new almost-deterministic code mentioned above). And secondly, in the setting of derandomizing linear space (rather than logarithmic space), the reconstruction procedure can afford to enumerate over all inputs (and can thus test each reconstructed circuit C_y to see how many inputs, if at all, C_y succeeds with). The proof appears in Section 6.4.

2.2 Minimal-Memory Derandomization From Weaker Assumptions

2.2.1 Minimal memory overhead from *nonuniform* assumptions

Our starting point is the (black-box) PRG construction of [DT23], constructed by composing two "low-cost" PRGs in order to get derandomization with minimal memory overhead (the composition idea, in the context of minimal *time* overhead derandomization, was already given in [CT21b]). In [DT23], the construction relied on two assumptions: High-end nonuniform hardness of a language in linear space, and (highly) space-efficient cryptographic PRGs with arbitrary polynomial stretch.

Specifically, given a probabilistic space-bounded machine $M(\cdot,\cdot)$, a cryptographic PRG G^{cry}, and the NW PRG NW^f based on the hard function f, the deterministic simulation of

M goes by enumerating over all seeds s to NW^f, and evaluating

$$\bar{M}(x, G^{\mathsf{cry}}(\mathsf{NW}^f)(s)),$$

where $\bar{M}$ is a variant of M that reads its random bits according to the current configuration, rather than in the standard order (see Lemma 7.1). The use of $\bar{M}$ instead of M is crucial: It allows saving an additional factor of S in the deterministic simulation space, where S is the overall space used by M (see Section 7.3).

Denoting by T the number of random bits read by M, the generator G^{cry} is used to decrease the number of random bits read by M to S^η for some small enough constant $\eta \in (0,1)$. Notice that fixing an input x, G^{cry} needs to fool the function $D(r) = \bar{M}(x, r)$. In [DT23], they were not able to utilize the fact that $M(x, \cdot)$ can be modeled as a read-once branching program, and so they resorted to using a cryptographic PRG that fools arbitrary circuits (in a sufficiently space-efficient manner). Indeed, the location of the next random bit read by $\bar{M}$ depends on the machine's own configuration, or in other words, depends *on the state of the ROBP* $M(x, \cdot)$. Moreover, no location is ever repeated twice.

We observe that this model precisely captures the notion of *adaptive order branching program*, so one can hope to use explicit tools from the space-bounded literature. Very recently, Chen, Lyu, Tal, and Wu [CLT+23] gave the first nontrivial PRG for that model. In fact, they show that the Forbes–Kelley PRG [FK18] is secure against this model and stretches $\mathrm{polylog}(T)$ bits into T bits, which comfortably works in our setting.

We prove that the Forbes–Kelley generator is also sufficiently efficient to take the place of the cryptographic PRG (Claim 7.3). Hence, combining with the result of [CLT+23], we are able to completely dispense with cryptographic assumptions (and replace G^{cry} with the [FK18] generator). We remark that the required explicitness property is *stronger* than that commonly used in the space-bounded derandomization literature, and several well-studied PRGs for ROBPs [Nis92; INW94] do not appear to obtain it. We also remark that this is the first use of PRGs for *generalizations* of ROBPs now studied in the literature, for the benefit of the *original* space-bounded derandomization question.

2.2.2 Minimal memory overhead from *uniform* assumptions

Next, similar to [DT23], we wish to establish the same minimal memory footprint derandomization result based on hardness of compression of multi-output functions. Given a multi-output function f, the deterministic simulation itself is similar to the one in Section 2.2.1, namely enumerating over

$$\bar{M}(x, G^{\mathsf{adp}}(\mathsf{NW}^{f(x)}(s))),$$

but now the NW generator uses the string $g = f(x)$ as the truth table of a hard function. The generator G^{adp} is the Forbes–Kelley generator, whereas in [DT23] it was a cryptographic one. Indeed, notice that our PRG is a *targeted* one, and thus the pseudorandom strings are chosen in a non-black-box (i.e., they depend on the input x). Also, as in previous works that employed targeted PRGs, and similar to the targeted somewhere-PRG of

Section 2.1, the analysis goes via an *instance-wise* hardness vs. randomness tradeoff: We show that if the derandomization fails on an input x, then a small-space machine succeeds in mapping the same x into a compressed version of $f(x)$.

The [DT23] reconstruction argument goes, very roughly, as follows. Given a distinguisher D for the composed PRG $G^{\mathsf{adp}} \circ \mathsf{NW}^g$, there is a *randomized* logspace algorithm that outputs, with high probability, a small circuit for g. This used a new reconstruction of the NW generator – a logspace-uniform $\mathbf{TC}^0$ one. Thus, under the assumption that there exists a function $f\colon \{0,1\}^n \to \{0,1\}^{n^2}$ that is hard to compress by randomized small-space algorithms for all but a finite number of inputs, [DT23] concludes that the composed PRG must be secure.

Here, we would like to weaken the hardness assumption and eliminate the randomness from the reconstruction procedure, which arises in two places. The first place is in the error correcting code, which could not be deterministically decoded in a space-efficient manner since it required polynomially-many random bits. Here, we can apply our new codes with deterministic decoding, described above in Section 2.1.2.

The second place that uses randomness relates to converting a distinguisher to a next-bit predictor. Indeed, we show that there exists a deterministic logspace reconstruction (for the NW generator that uses our new code), that given a *next-bit predictor* P, prints a small oracle $\mathbf{TC}^0$ circuit C such that C^P computes the hard function g. Typically in reconstruction arguments, one can simply argue that given a distinguisher that breaks the PRG, a suitable next-bit predictor *exists*, as long as we don't care about uniformity. Also, if the distinguisher is itself an ROBP, we can (unconditionally) transform it into a next-bit predictor in logspace, deterministically. In our case, however, the distinguisher for the NW PRG is $D \circ G^{\mathsf{adp}}$, where $D(r) = \bar{M}(x, r)$ as above. This distinguisher is *not* computable by an ROBP, an so we cannot perform the reconstruction as-is. This is where Assumption 2 enters the picture. We show that if we have an HSG that hits $(\mathbf{TC}^0)^{D \circ G^{\mathsf{adp}}}$, we can use it to find a string that "completes the hybrid argument", and use this string to obtain a next-bit predictor in a space-efficient manner (see Lemma 7.6). We leave the rest of the details to Section 7.3.

2.3 A New Proof of $\mathbf{BPL} \subseteq \mathbf{CL}$

Finally, we use our techniques to study the relationship between randomized and *catalytic* computation. Recall that in the catalytic logspace model we have $c \log n$ bits of working memory, and a catalytic tape w of length n^c that has an arbitrary initial configuration. We can modify this tape arbitrarily, but we must reset it to its original configuration at the end of the computation.

There are two known proofs establishing that $\mathbf{BPL} \subseteq \mathbf{CL}$. The first, which follows from [BCK+14], uses algebraic techniques involving reversible computation over a ring (and ultimately proves the stronger result that logspace uniform $\mathbf{TC}^1 \subseteq \mathbf{CL}$). The second (see [Mer23]) follows a *compress-or-random* approach. In particular, it treats the catalytic tape as a sequence of random walks. For an ROBP B that we wish to derandomize, either this set of walks fools B (in which case we can use them to derandomize), or there is some state in the ROBP at which the distribution of outgoing walks is skewed (in which case one can use an in-place compression algorithm to free up many bits on the catalytic tape, which can then be used as the workspace for a space-inefficient derandomization algorithm).

We give a new approach, that is likewise based on the compress-or-random dichotomy, but is more modular. We treat the catalytic tape $\mathbf{w}$ as a candidate *hard truth table*, and instantiate (a version of) the NW generator NW with the table $\bar{\mathbf{w}}$, where $\bar{\mathbf{w}}$ is the encoding of $\mathbf{w}$ using the code of Section 2.1.2. We use a version of the NW generator that has deterministic reconstruction, where given a previous-bit-predictor for $\mathsf{NW}^{\bar{\mathbf{w}}}$ we can reconstruct in deterministic logspace a noisy version of $\bar{\mathbf{w}}$. Furthermore, as our code has deterministic decoding, we use this noisy version of $\bar{\mathbf{w}}$ to approximately decode $\mathbf{w}$ in logspace.

Then, given a branching program B, we apply the distinguish-to-predict transformation for ROBPs (Theorem 4.2) to B, obtaining a family of candidate predictors $P_1, \ldots, P_{\mathrm{poly}(n)}$. Next, we test if $\mathsf{NW}^{\bar{\mathbf{w}}}$ is predicted with non-negligible advantage by any such P_i. We then break into cases:

1. If no such P_i predicts $\mathsf{NW}^{\bar{\mathbf{w}}}$ with non-negligible advantage, we have that $\mathsf{NW}^{\bar{\mathbf{w}}}$ fools B, and so we can derandomize without writing to the catalytic tape.

2. If there is $P = P_i$ that predicts $\mathsf{NW}^{\bar{\mathbf{w}}}$ with non-negligible advantage, we use the deterministic reconstruction and decoding algorithms applied with P to obtain a small circuit C such that $C(j) = \mathbf{w}_j$ on the vast majority of indices j. We then identify a large interval I of $\mathbf{w}$ such that $C(j) = \mathbf{w}_j$ for every $j \in I$. Thus, as long as we can retain the ability to evaluate C, we can *erase* $\mathbf{w}_I$ and thus use $\mathbf{w}_I$ as the workspace for a space-inefficient derandomization algorithm (e.g. Nisan's [Nis94]). However, we must we careful that even after erasing we maintain the ability to evaluate the circuit C on $j \in I$, and hence reset the tape to its original configuration.

   To evaluate C, it suffices to have access to P (which we can retain by remembering the short description of the predictor)[16] and sub-linearly many bits of $\bar{\mathbf{w}}$ (i.e., the encoded version of $\mathbf{w}$). Naively, of course, each bit of $\bar{\mathbf{w}}$ could depend on all bits of $\mathbf{w}$, making erasing impossible. To avoid this, we use the *locally encodable* version of our code (Theorem 5.2).[17] The full details are given in Section 8.

3 Preliminaries

Strings and distributions. Given $x \in \{0,1\}^n$, let $x_{<i} = x_{1\ldots i-1}$, $x_{\leq i} = x_{1\ldots i}$, $x_{>i} = x_{i+1\ldots n}$, $x_{\geq i} = x_{i\ldots n}$ and let $x_{<1}$ and $x_{>n}$ be the empty string. For an integer n, we denote $[n] = \{1, \ldots, n\}$. Given a set S, let U_S be the uniform distribution over the set S, and for $n \in$

[16] Here, we crucially use the fact that our new distinguisher-to-predictor transformation from Theorem 4.2 outputs a description of only logarithmic size.

[17] This is why we only approximately decode, but this suffices for our application.

$\mathbb{N}$ let $U_n = U_{\{0,1\}^n}$. We say that a distribution D ε-fools a function $f\colon \{0,1\}^n \to \{0,1\}$ if $|\mathbb{E}[f(D)] - \mathbb{E}[f(U_n)]| \leq \varepsilon$. In addition to distinguishers, we will also need next-bit predictors.

Definition 3.1 (distinguisher)**.** *We say that $T\colon \{0,1\}^n \to \{0,1\}$ is an ε-distinguisher for a distribution D over $\{0,1\}^n$ if*

$$\left| \Pr_{r \leftarrow U_n}[T(r) = 1] - \Pr_{x \leftarrow D}[T(x) = 1] \right| > \varepsilon \,.$$

Definition 3.2. *We say that $P : \{0,1\}^i \to \{0,1\}$ is an ε-*next-bit predictor *(resp. ε-*previous-bit predictor*) for a distribution D over $\{0,1\}^n$ if $\Pr_{x \leftarrow D}[P(x_{\leq i}) = x_{i+1}] \geq 1/2 + \varepsilon$ (resp. $\Pr_{x \leftarrow D}[P(x_{>n-i}) = x_{n-i}] \geq 1/2 + \varepsilon$).*

Definition 3.3 (PRGs and HSGs)**.** *We say that $G\colon \{0,1\}^s \to \{0,1\}^n$ is an ε-*pseudorandom generator (PRG) *for a class of functions $\mathcal{F}\colon \{0,1\}^n \to \{0,1\}$ if for every $f \in \mathcal{F}$, $G(U_s)$ fools f, i.e.*

$$|\mathbb{E}[f(U_n)] - \mathbb{E}[f(G(U_s))]| \leq \varepsilon.$$

*We say that G is an ε-*hitting set generator *(HSG) if for every $f \in \mathcal{F}$ such that $\mathbb{E}[f(U_n)] \geq \varepsilon$, there exists $z \in \{0,1\}^s$ such that $f(G(z)) = 1$.*

3.1 Space Bounded Computation, and Branching Programs

We use the standard model of space-bounded computation (see also [Gol08, Section 5] or [AB09, Section 4]). A deterministic space-bounded Turing machine has three semi infinite tapes: an *input tape* (that is read-only); a *work tape* (that is read/write) and an *output tape* (that is write-only and uni-directional). The machine's alphabet is $\{0,1\}$. The space complexity of the machine is the number of used cells on the work tape. We say that a language is in $\mathsf{DSPACE}[s(n)]$ if it is accepted by a space bounded TM with space complexity $s(n)$ on inputs of length n. Naturally, space-bounded machines can also compute *functions* on the output tape.

A *probabilistic* space-bounded Turing machine is similar to the deterministic machine except that it can also toss random coins. We also require a space-$s(n)$ probabilistic machine to always halts within $2^{s'(n)}$ steps, where $s'(n) = s(n) + O(\log s(n)) + \log n$ is the number of possible configurations.[18] Note that this bound on the runtime always holds for (halting) space-$s(n)$ deterministic machines.

One convenient way to formulate this is by adding a fourth semi-infinite tape, the random-coins tape, that is read-only, uni-directional and is initialized with perfectly uniform bits. We are concerned with bounded-error computation: We say a language is accepted by a probabilistic Turing machine if for every input in the language the acceptance probability is at least $2/3$, and for every input not in the language it is at most $1/3$.

[18]The machine's configuration includes the content of its work tapes, its current state, and the location of its heads, including the head on the input tape. For convenience, we can assume that the heads location and current state are written on dedicated worktapes.

Similarly, we denote by $\mathsf{BPSPACE}[s(n)]$ the set of languages accepted by a probabilistic space-bounded TM with space complexity $s(n)$.

On multi-tape machines. While we defined the space bounded complexity class with respect to a single work tape, throughout the paper we often describe computations done on multiple work tapes. As long as the number of work tapes is some universal constant, which will indeed be the case, the simulation loss is negligible and we will ignore it. Formally, it follows from the following simple observation.

Claim 3.4. *Let M be a (deterministic or probabilistic) space-bounded TM with $C > 1$ work tapes, such that on input of length n uses space $s = s(n) \geq \log n$ (in total over all its work tapes). Then, M can be simulated by a TM with a single work tape that uses $s + O(C \cdot \log s)$ space.*

Composition of space-bounded algorithms. We will heavily use space-efficient composition of functions computable by space-bounded TMs.

Proposition 3.5 ([Gol08], Lemma 5.2). *Let $f_1, f_2 \colon \{0,1\}^\star \to \{0,1\}^\star$ be functions that are computable in space $s_1, s_2 \colon \mathbb{N} \to \mathbb{N}$. Then, $f_2 \circ f_1 \colon \{0,1\}^\star \to \{0,1\}^\star$ can be computed in space*

$$s(n) = s_2(\ell_1(n)) + s_1(n) + \log(\ell_1(n)) + O(1)$$

where $\ell_1(n)$ is a bound on the output length of f_1 (i.e., the cells used on the work tape) on inputs of length n.

We note that the bound in Proposition 3.5 assumes two work tapes, and as we stated above, simulating $f_2 \circ f_1$ on a single work tape incurs an additional $O(\log s(n))$ additive factor in space.

When we say that a function $f \colon \{0,1\}^n \to \{0,1\}^m$, which can be viewed as $f \colon \{0,1\}^n \times [m] \to \{0,1\}$, is *logspace computable* if it is computable in space $O(\log n + \log\log m)$. When we compute a function using an *oracle* machine, we account for the space needed to prepare the input to the oracle (unless stated otherwise, we write the entire input to the tape).

Branching programs. We recall two models of read-once branching program: read-once branching programs (also known as standard-order branching programs), and (read-once) adaptive order branching programs.

Definition 3.6 (ROBP). *A* read-once branching program (ROBP) *B of width w and length n is specified by an initial state $v_{st} \in [w]$, an accepting state $v_{ac} \in [w]$ and a sequence of transition functions $B_i \colon [w] \times \{0,1\} \to [w]$ for $i \in [n]$. The ROBP naturally defines a function $B \colon \{0,1\}^n \to \{0,1\}^n$: Start at v_{st}, and then for $i = 1, \ldots, n$, read the input symbol x_i and transition to the state $v_i = B_i(v_{i-1}, x_i)$. The ROBP accepts x, i.e., $B(x) = 1$, if and only if $v_n = v_{ac}$.*

In the *adaptive* read-once model, each computation path of the branching program can read the bits of input $r \in \{0,1\}^n$ in a different order, as long as each bit is read exactly once.

Definition 3.7 (AOBP). *A (read-once)* adaptive order branching program (AOBP) *B of width w and length n, is a layered 2-out-regular directed graph with $n+1$ layers, each layer having w vertices, which is also equipped with a labeling function $l\colon V \to [n]$ where V denotes the set of vertices of B, and includes a start and accept vertices v_{st}, v_{ac}.*

The AOBP defines a function $B\colon \{0,1\}^n \to \{0,1\}$ as follows. Start at v_{st}, and then for $i = 1, \dots, n$, transition to the state $v_i = B(v_{i-1}, x_{l(v_{i-1})})$, where $B(u,b)$ denotes the σ^{th} neighbor of u in B. The AOBP accepts x, i.e., $B(x) = 1$, if and only if $v_n = v_{ac}$. Moreover, we require that for every possible input $x \in \{0,1\}^n$, every bit of x is read at most once over the computation.

When we refer to the *size* of a branching program, we mean the number of vertices of the underlying layered directed graph, namely $(n+1) \cdot w$.

3.2 Circuits and Hardness Notions

We will use the standard definitions of circuit classes. In particular, an AC^i circuit is a Boolean circuit with depth $O(\log^i n)$ over the De Morgan basis with unbounded fan-in gates. In TC^i, we allow Majority gates in addition to NOT, OR, and AND, but we will sometimes allow threshold gates as an intermediate model. We define the *size* of the circuit to be its number of wires. We say that an oracle circuit is *non-adaptive* if each computation path contains at most one oracle call. Throughout the paper, we fix the following standard way of describing circuits as strings. Specifically, the description consists of a list of gates, where the description of each gate consists of its type (i.e., the function that it computes) and of the indices of gates that feed into it. Observe that the description length of a circuit with s gates and $w \geq s$ wires is $O(w \log s)$.[19]

We say that a family of circuits $\{C_n\}$, each C_n is of size $s(n)$, is *logspace uniform* if there exists a deterministic algorithm that runs in space $O(\log(s(n))$ and outputs the description of C_n. We will sometimes use stronger notions of uniformity which we will define along the way.

We say that a function f (more accurately, a family of functions) is *hard* for a class $\mathbf{C}$ if no function from $\mathbf{C}$ correctly computes f, up to perhaps a finite number of inputs lengths.

We will use the following standard notion of average-case hardness, which asserts that $L \in \mathsf{avg}_\varepsilon \mathcal{F}$ if there is an algorithm $f \in \mathcal{F}$ deciding L on $1-\varepsilon$ of the inputs and with *zero-error* (cf., [IW98; BT06]).

Definition 3.8. *For a class $\mathcal{F}$ of functions $\{0,1\}^\star \to \{0,1\} \cup \{\perp\}$, and $\varepsilon > 0$, we say that a language $L \in \mathsf{avg}_\varepsilon \mathcal{F}$ if there exists $f \in \mathcal{F}$ such that for every sufficiently large $n \in \mathbb{N}$:*

1. $\Pr_{x \leftarrow U_n}[f(x) = L(x)] \geq 1 - \varepsilon$, *and,*
2. *For every $x \in \{0,1\}^n$ we have $f(x) \in \{L(x), \perp\}$.*

[19] In Section 6, where we consider weighted threshold gates, the description size will be $O(w(t + \log s))$, for t being the number bits required to write each weight or threshold, but this will essentially be the same bound.

Next, we make the notion of circuits with oracle to complexity classes precise. For a circuit class $\mathbf{C}$ (say, TC^0) and a class of languages $\mathcal{L}$, we let $\mathbf{C}^{\mathcal{L}}$ be the following class of languages. We say that $L \in \mathbf{C}^{\mathcal{L}}$ if there exists $A \in \mathcal{L}$ and a family of $\mathbf{C}$-circuits that decides L, say $\{C_n\}_{n\in\mathbb{N}}$, such that each C_n can have oracle queries to A. When we parameterize $\mathcal{L}$ by input lengths, say $\mathsf{DSPACE}[n]$, we treat n as the *queries length* rather than the length of the input to the circuit. Finally, for a circuit C mapping n bits to one bit, we denote by $\mathsf{tt}(C)$ the truth-table of the corresponding function, namely the length-2^n string for which $\mathsf{tt}(C)_x = C(x)$.

3.3 Error Correcting Codes

We say that an error correcting code $\mathcal{C}\colon \Sigma^k \to \Sigma^n$ has *relative distance* δ if for any distinct codewords $x, y \in \mathcal{C}$, it holds that $\delta(x,y) = \Pr_{i\in[n]}[x_i \neq y_i] \leq \delta$. As customary, we often use $\mathcal{C}$ to simply denote $\mathrm{Im}(\mathcal{C}) \subseteq \Sigma^n$. If one corrupts a codeword in less than $\delta/2$ fraction of its coordinates, *unique decoding* is possible. Otherwise, one can resort to *list decoding*. We say that $\mathcal{C}$ is (ρ, L) list decodable if for any $w \in \Sigma^n$ there are at most L codewords $c \in \mathcal{C}$ that satisfy $\delta(w,c) \leq 1-\rho$. We refer to ρ as the *agreement* parameter.

We will be interested in the *local* variants of unique and list decoding, wherein the algorithmic task of decoding a single coordinate can be done very efficiently. Moreover, we will sometimes need the *approximate* variant, in which we allow the returned words to only agree with some corresponding codewords in a large fraction of the coordinates.

Definition 3.9 (locally approximately list-decodable code). *We say that a code $\mathcal{C}\colon \Sigma^k \to \Sigma^n$ is* locally approximately list decodable *from agreement ρ to agreement $1-\delta$, with Q queries, by circuits of size s and list size L, if there exist randomized circuits $\mathsf{Dec}_1, \dots, \mathsf{Dec}_L$, each of size s, that satisfy the following.*

- *Each Dec_i has oracle access to a received word $r \in \Sigma^n$, and makes at most Q queries to the coordinates of r.*
- *For every $r \in \Sigma^n$, and $c = \mathcal{C}(x)$ that agrees with r in at least ρ-fraction of its coordinates, there exists $j \in [L]$ such that*

$$\Pr_{i\in[n]}\left[\Pr_{\mathsf{Dec}_j}\left[\mathsf{Dec}_j^r(i) = x_i\right] \geq \xi\right] \geq 1-\delta,$$

  for some error parameter $\xi > 0$.

When $\delta = 0$, we say that $\mathcal{C}$ is locally list decodable. When $L = 1$, we say that $\mathcal{C}$ is locally (approximately) uniquely decodable.

When we do not pose any uniformity constraints, the output list size parameter L may only implicit, in the sense that each Dec_i is of size at least $\log L$ and we will sometimes omit it from the above notation. Similarly, when we do not insist on a uniform generation of the Dec_i-s, by standard error reduction, we can take $\xi = 1$ and incur only a minor loss in parameters. Often, we will require additional properties from our list-decodable codes, and we will define those properties explicitly when needed.

3.4 Pseudorandomness Primitives

Samplers. We recall the definition of a (strong) sampler:

Definition 3.10 (strong sampler)**.** *A function* $\mathsf{Samp}\colon \{0,1\}^m \times [t] \to \{0,1\}^n$ *is a* strong (ε, δ) (oblivious) sampler *if for any* $H_1, \ldots, H_t \subseteq \{0,1\}^n$ *it holds that*

$$\Pr_{x \leftarrow \{0,1\}^m}\left[\left|\Pr_{i \leftarrow [t]}[\mathsf{Samp}(x,i) \in H_i] - \mathop{\mathbb{E}}_{i \leftarrow [t]}[\rho(H_i)]\right| \le \varepsilon\right] \ge 1 - \delta,$$

where we denote by $\rho(H_i) = \frac{|H_i|}{2^n}$ *the density of a set.*

The parameter ε is the *accuracy* parameter of the sampler, and δ is its *confidence* parameter. We recall the sampler of [Hea08], as it gives a sampler with parameters matching that of the expander walk sampler, combined with good explicitness properties.

Theorem 3.11 ([Hea08, Theorem 1.3])**.** *For every* $n \in \mathbb{N}$ *and any* $\varepsilon, \delta > 0$ *there exists a strong* (ε, δ) *sampler* $\mathsf{Samp}\colon \{0,1\}^m \times [t] \to \{0,1\}^n$ *where* $t = O(\log(1/\delta)/\varepsilon^2)$ *and* $m = n + O(t)$. *Moreover, there is a space* $O(\log m)$ *algorithm that outputs an* $\mathbf{AC}^0[\oplus]$ *circuit of size* $\mathrm{poly}(m)$ *that computes* Samp. *In particular, given* $x \in \{0,1\}^m$ *and* $i \in [t]$, $\mathsf{Samp}(x,i)$ *is computable in space* $O(\log m)$.

We also recall the following strong sampler, that has better randomness complexity at the expense of worse sampling complexity.[20]

Theorem 3.12 ([Gol11a; CL20])**.** *For every* $n \in \mathbb{N}$ *and* $\varepsilon, \delta > 0$, *there exists an explicit strong* (ε, δ) *sampler* $\mathsf{Samp} : \{0,1\}^m \times [t] \to \{0,1\}^n$ *where* $t = \mathrm{poly}(\log(1/\delta)/\varepsilon)$ *and* $m = n + O(\log(1/\varepsilon\delta))$. *Moreover, given* $x \in \{0,1\}^m$ *and* $y \in [t]$, $\mathsf{Samp}(x,y)$ *is computable in space* $O(m)$.

Small-bias sets. We likewise recall the definition of small-bias spaces.

Definition 3.13. *A function* $G\colon \{0,1\}^t \to \{0,1\}^k$ *is an* ε-biased generator *if* $G(U_t)$ *is a* ε*-biased probability space over* $\{0,1\}^k$, *which formally means that for every* $T \in \{0,1\}^k$,

$$\Pr_{y \leftarrow U_t}[\langle T, G(y)\rangle = 1] \in [1/2 - \varepsilon, 1/2 + \varepsilon].$$

We recall that strongly explicit small-bias spaces exist with asymptotically optimal seed length. Moreover, by [HV06], these spaces can be computed in space logarithmic in the seed.

Proposition 3.14 ([NN93; HV06])**.** *Given* $k \in \mathbb{N}$ *and* $\varepsilon > 0$, *there is an* ε*-biased generator* $\mathsf{Bias}\colon \{0,1\}^t \to \{0,1\}^k$ *with seed length* $t = O(\log(k/\varepsilon))$. *Moreover, the transformation that maps* $(x,j) \in \{0,1\}^t \times [k]$ *to* $\mathsf{Bias}(x)_j$ *can be computed in space* $O(\log t)$.

[20] We note that the "strongness" property does not appear in [RVW02; Gol11a; CL20] (the standard, non-strong, definition assumes $H_1 = \ldots = H_t$). However, the seeded extractor that is used to construct the sampler can be made strong with essentially no loss in parameters, and strong extractors yield strong samplers (see [Zuc97]).

Designs.

Definition 3.15 (combinatorial design)**.** *A family of sets $S_1, \dots, S_k \subset [d]$ is called an (n, a)* combinatorial design *if each of the sets is of size $|S_i| = n$, and any distinct sets S_i, S_j satisfy $|S_i \cap S_j| \leq a$. The corresponding function* $\mathsf{Des}\colon \{0,1\}^d \times [d] \to \{0,1\}^n$ *takes as input $z \in \{0,1\}^d$ and $i \in [k]$ and outputs the restriction of z to the coordinates in S_i.*

We recall that logspace-uniform designs exist with good parameters:

Theorem 3.16 ([KM02], Lemma 5.19)**.** *There is a universal constant $c \geq 1$ such that for any $\alpha \in (0,1)$ the following holds for sufficiently large n. There is an algorithm that outputs an $(n, \alpha n)$ design $S_1, \dots, S_k \subset [d]$ where $k = \lceil 2^{(\alpha/c)n} \rceil$ and $d \leq (c/\alpha)n$. On input $i \in [k]$, this algorithm runs in space $O(n)$ and outputs S_i. In particular, the corresponding function* $\mathsf{Des}\colon \{0,1\}^d \times [k] \to \{0,1\}^n$ *is computable in space $O(n)$.*

k-wise independence. We say that $Z \sim \{0,1\}^n$ is k-wise independent is for any $I = \{i_1, \dots, i_k\} \subseteq [n]$ it holds that $Z|_I = U_{|I|}$. It is well-known that one can efficiently sample from a k-wise independent distribution over $\{0,1\}^n$ using $O(k \log n)$ bits. One way to do so is by evaluations of random polynomials of degree $k-1$. This gives a space-efficient way of sampling from Z.

Claim 3.17. *For any integers n and $k \leq n$, there exists a k-wise independent distribution over $\{0,1\}^n$. Denoting by $S = \{s_1, \dots, s_m\}$ its support size, we have that $m = n^k$, and the transformation that maps $(i,j) \in [m] \times [n]$ to $s_i[j]$ can be computed in space $O(\log k + \operatorname{loglog} n)$.*

To see that, assume without loss of generality that n is a power of 2 and let $\mathbb{F}$ be a field of size n. Then, $s_i[j]$ can be computed by:

- Using $i \in [n^k]$ to sample coefficients $(a_0, \dots, a_{k-1}) \in \mathbb{F}^k$,
- Compute $\sum_{\ell=0}^{k-1} a_\ell j^\ell$ over $\mathbb{F}$, where we interpret j as an element of $\mathbb{F}$, and,
- Taking any field trace from $\mathbb{F}$ to $\mathbb{F}_2$.

Under the standard representation of $\mathbb{F}$, and composition of space-bounded algorithms (Proposition 3.5), the above can be done in space $O(\log k + \log\log n)$.[21]

[21] Iterated addition and multiplication can be done by logspace-uniform (and even logtime uniform) TC^0 circuits so in particular in logspace. More concretely, adding and multiplying k $\mathbb{F}$-elements can be done by $\operatorname{poly}(k \log n)$-sized TC^0 circuits [HAB02], so in particular in space $O(\log k + \operatorname{loglog} n)$. One can do even better, since computing a field element of the k^{th} power can be done in size $\operatorname{poly}(\log k, \log n)$ for specific representations of $\mathbb{F}$ [HV06] but we won't need this fact (recall that iterating over the $k \log n$ bits of i already takes $\log(k \log n)$ space).

3.5 Catalytic Computation

Catalytic computation, defined by Buhrman *et al.* [BCK+14] (see also [BKL+16]) asks whether an auxiliary memory, that already stores some data that should be restored for later use, can be useful for computation. That is, can we make computations more efficient if in addition to a standard clean worktape, we have access to additional space which is initially in an arbitrary state and must be returned to that state when our computation is finished?

Formally, we enrich our model of (deterministic) space-bounded Turing machines with an auxiliary tape, which we call the *catalytic tape*. For every possible initial setting of the catalytic tape, at the end of the computation the Turing machine must have returned the tape to its initial contents. We denote by $\mathbf{CSPACE}[s(n), s_A(n)]$ to be the set of all languages that can be decided by a catalytic TM that runs in (standard) space $s(n)$ and uses $s_A(n)$ cells of the auxiliary tape. Clearly, a catalytic TM can compute a function (in working space s and catalytic space s_A) of the input, rather than simply outputting accept or reject.

4 An Improved Local Consistency Test

Existing local consistency tests [Nis93; CH22; GRZ23; PRZ23] have at least one of several undesirable properties: either they output a distinguisher, not a next-bit-predictor; or the construction of the predictor requires a large amount of advice; or the soundness loss is large (in particular, it depends on the width of the program, not merely the length).

These properties are prohibitive for our applications. Thus, we rectify this situation, by giving a very simple test that simultaneously:

- Can be described very efficiently given B.
- Obtains soundness loss exactly matching the (non-explicit) hybrid argument.
- Outputs a next-bit predictor.

To do so, instead of outputting a next-bit-predictor, we output a *previous*-bit predictor instead; that is, we obtain a program that reads the *last* k bits of the output of a generator and predicts the *preceding* $(n - k - 1)^{\text{st}}$ bit. Interestingly, only know how to obtain a test that does not suffer from the undesirable properties above by outputting such a previous-bit-predictor. We do so by an analysis that uses a hybrid argument performed in the backwards direction (i.e., in reverse direction compared to the standard transformation from distinguishability to predictability).

To define the test, we first introduce notation for subprograms of an ROBP.

Definition 4.1. *For an ROBP $B\colon \{0,1\}^n \to \{0,1\}$ of length n and width w, let $B_{i,j}$ be the subprogram of length $n - i$ and width w defined as follows. We let $B_{i,j}$ be B with the first i layers removed, and vertex j in layer i marked as the new start vertex. Note that $B_{i,j}$ can be described with advice $\log nw$ given B, and can be constructed in logspace given B.*

Theorem 4.2. *Given an ROBP B of length n and width w, for every $i \in [n], j \in [w], b \in \{0,1\}$, let $P_{i,j,b} \colon \{0,1\}^{n-i} \to \{0,1\}$ be defined as $P_{i,j,b}(x) = B_{i,j}(x) \oplus b$. Then for every $\delta > 0$, for every ROBP B, for every distribution D over $\{0,1\}^n$, at least one of two events occurs:*

1. $|\mathbb{E}[B(D)] - \mathbb{E}[B(U_n)]| \le \delta$*, or,*
2. *There is i, j, b such that* $\Pr_{x \leftarrow D}[P_{i,j,b}(x_{>i}) = x_i] > \frac{1}{2} + \frac{\delta}{n}$.

Proof. First, assume there is no such (i, j, b). We now assume that Item 1 does not occur and derive a contradiction. For $i \in \{0, \dots, n\}$ let $Z_i = (U_i \circ D_{>i})$. By assumption, we have $|\mathbb{E}[B(Z_n))] - \mathbb{E}[B(Z_0)]| > \delta$.

By the standard transformation from distinguishability to predictability, there is $z \in \{0,1\}^i$ and $b \in \{0,1\}$ such that

$$\Pr_{x \leftarrow D}[B(z \circ x_{>i}) \oplus b = x_i] > \frac{1}{2} + \frac{\delta}{n}.$$

But observe that $B(z \circ x_{>i}) = B_{i,j}(x_{>i})$ for some $j \in [w]$, as fixing the first i bits to z is equivalent to starting the computation from state $j = B[v_{st}, z]$, and hence

$$\Pr_{x \leftarrow D}[P_{i,j,b}(x_{>i}) = x_i] > \frac{1}{2} + \frac{\delta}{n},$$

contradicting our assumption. ∎

We note that in some cases one would like the tests that verify if a PRG is good for B to be themselves implementable by ROBPs. This is direct in the case that the tests are next-bit-predictors at some vertex of B. In this case, we can still achieve it at the cost of a factor of two in the width, by creating the program that stores x_i, computes $t = B_{i,j}(x_{>i})$, and accepts if $x_i = b \oplus t$. Such a program has expectation exactly $1/2$ under the uniform distribution, and will have expectation far from $1/2$ for some i, j, b for any bad PRG.

5 Local List Decoding in Deterministic Logspace TC^0

In this section we will construct our locally list decodable code, decodable by *deterministic* uniform TC^0 circuits, generated by space-efficient algorithms that take only a small seed.

The construction follows along the lines of [DT23], but here we will need to keep track of (and save upon) the randomness we use (similar to what is done in [PRZ23]), and the uniformity of the decoding and encoding, among other modifications that will be explained later on.

At a high level, our main code $\mathcal{C}$, given in the theorem below, is a composition of a (variant of) the "GGHKR code" [GGH+07] with the IW derandomized direct product code [IW97], concatenated with the Hadamard code. In Section 5.1 we will describe the current work's implementation of GGHKR, which is a bit more involved than the one in [DT23]. In Section 5.2 we will establish the encoding and decoding properties we need from IW and Had.

Theorem 5.1 (the code $\mathcal{C}$)**.** *There exists a family of logspace-computable codes*

$$\mathcal{C}\colon \{0,1\}^k \to \{0,1\}^n,$$

such that for any $\varepsilon > 0$, any confidence $\delta > 0$ and any constant $\gamma > 0$, we have that $n = \mathrm{poly}\big((k/\varepsilon)^{1/\gamma}\big)$, and the following holds for some universal constant $c > 1$.

- ***Local List Decoding.*** *There exists an oracle algorithm* Dec *that runs in space $S_{\mathsf{Dec}} = O_\gamma(\log(k/\varepsilon\delta))$, takes as input a seed y of length $O(\frac{1}{\gamma}\log\frac{k}{\varepsilon\delta})$, an advice $j \in [L]$ for $L = \widetilde{O}(\log(1/\delta)/\varepsilon^2)$, and makes at most*

$$Q_{\mathsf{Dec}} = \left(\frac{k^\gamma \log(1/\delta)}{\varepsilon}\right)^c$$

  non-adaptive queries to its oracle, such that $\mathsf{Dec}^x(y,j)$ outputs a (deterministic) oracle TC^0 circuit $C_{y,j}$ of size $s = \widetilde{O}(Q_{\mathsf{Dec}})$, so that for any $w \in \{0,1\}^n$, if $x \in \{0,1\}^k$ is such that $c = \mathcal{C}(x)$ agrees with w in at least $1/2 + \varepsilon$ fraction of its coordinates, then

$$\Pr_y\big[\exists j \in [L], \forall i \in [k]\ C^w_{y,j}(i) = x_i\big] \geq 1 - \delta.$$

- ***Non-adaptivity.*** *Both* Dec *and each $C_{y,j}$ are non-adaptive. In particular,*
  - *On input (y,j), there exists a space-S_{Dec} deterministic algorithm that outputs a list $\mathcal{L}_{\mathsf{Dec}}(y,j) \subseteq [k]$ of size Q_{Dec} such that $\mathsf{Dec}^x(y,j)$ only ever queries x in locations in $\mathcal{L}_{\mathsf{Dec}}(y,j)$.*
  - *Each $C_{y,j}$ makes at most $Q = \mathrm{polylog}(k/\delta) \cdot \mathrm{poly}(1/\varepsilon)$ non-adaptive queries to w. Consequently, on input (y,j), there exists a space-S_{Dec} deterministic algorithm that outputs $\mathcal{L}_C(y,j) \subseteq [n]$ of size $k \cdot Q$ such that $C^w_{y,j}(i)$ only ever queries the received word at locations in $\mathcal{L}_C(y,j)$.*

We will also need a variant of $\mathcal{C}$ that is also *locally encodable*. Naturally, this comes at the expense of (perfect) local decodability, and so our code will be *approximately* locally list decodable. Technically, this is achieved by modifying the GGHKR code, and we elaborate on it in Section 5.1. We also need our locally encodable code $\mathcal{C}_{\mathsf{LE}}$ to be encodable in uniform TC^0, and not only in logarithmic space.

Theorem 5.2 (the code $\mathcal{C}_{\mathsf{LE}}$)**.** *There exists a family of logspace-computable, systematic*[22]*, codes*

$$\mathcal{C}_{\mathsf{LE}}\colon \mathbb{F}^k \to \{0,1\}^n,$$

parameterized by $d \leq k$, such that for any $\varepsilon > 0$, any confidence $\delta > 0$ and any constant $\gamma > 0$, we have that $n = \mathrm{poly}\big((k/\varepsilon)^{1/\gamma}\big)$, and the following holds for some universal constants $c > 1$ and $c' \in (0,1)$, as long as $|\mathbb{F}|$ is at most exponential in k.

[22] We use a slightly weaker notion of systematic codes than the standard definition; see Claim 5.25 for the details.

- ***Approximate Local List Decoding.*** *There exists an algorithm* Dec *that runs in space $O_\gamma(\log(k/\varepsilon\delta))$, takes as input a seed y of length $O(\frac{1}{\gamma}\log\frac{k}{\varepsilon\delta})$ and an advice $j \in [L]$ for $L = \widetilde{O}(\log(1/\delta)/\varepsilon^2)]$, and outputs a (deterministic) oracle TC^0 circuit $C_{y,j}$ of size*

$$s_{\mathsf{Dec}} = \left(\frac{k^\gamma d \log(|\mathbb{F}|/\delta)}{\varepsilon}\right)^c,$$

  so that for any $w \in \{0,1\}^n$, if $x \in \mathbb{F}^k$ is such that $\mathcal{C}(x)$ agrees with w in at least $1/2+\varepsilon$ fraction of its coordinates, then

$$\Pr_y\left[\exists j \in [L], \Pr_{i \leftarrow [k]}\left[C^{x,w}_{y,j}(i) = x_i\right] \geq 1 - d^{-c'}\right] \geq 1 - \delta.$$

  We stress that the depth of each $C_{y,j}$ is a universal constant, and in particular independent of γ.

- ***Non-adaptivity.*** *Each $C_{y,j}$ comprises two circuit.*
  - *A top* preprocessing *circuit that makes at most $Q_{\mathsf{pre}} = \mathrm{poly}(k^\gamma, 1/\varepsilon, \log(1/\delta))$ non-adaptive queries to the message x, independent of i, and,*
  - *A bottom* decoder, *that gets i, and the queried coordinates of x, and makes at most $Q = \mathrm{polylog}(k/\delta) \cdot \mathrm{poly}(1/\varepsilon) \cdot \widetilde{O}(d^2 \log|\mathbb{F}|)$ non-adaptive queries to the corrupt word w.*

- ***Local Encoding in Uniform Constant-Depth.*** *The encoding map has locality*

$$D = d \cdot \mathrm{poly}(1/\varepsilon, \log(1/\delta)).$$

  Moreover, there is a logspace-uniform TC^0 circuit of size $\mathrm{poly}(d, \log k, 1/\varepsilon, \log(1/\delta))$, that on input $i \in [n]$, returns the D coordinates $q_1, \ldots, q_D$ to be queried, and another TC^0 circuit of size $\mathrm{poly}(d, \log k, \log|\mathbb{F}|, 1/\varepsilon, \log(1/\delta))$, generated in space $O(\log(k/\varepsilon))$, that given $x_{q_1}, \ldots, x_{q_D} \in \mathbb{F}$, outputs $\mathcal{C}_{\mathsf{LE}}(x)_i$.

  Here too, the constant depth of the encoding circuits is independent of γ.

Remark 5.3. We suspect that the decoding space complexity's dependence on δ in Theorems 5.1 and 5.2 is not optimal. For example, if we could replace the sampler in the proof of Lemma 5.19 by a sampler whose space complexity is $O(\mathrm{loglog}(1/\delta))$ (such as the median-of-averages sampler, see [BGG93; CDS+23]), then the space complexity of the entire construction might be doubly logarithmic in $1/\delta$ (the randomness complexity in this case would increase by an additive factor of $\log(1/\delta) \cdot \mathrm{loglog}(n)$). We did not try to optimize this dependency, since in this paper we only use the codes with a constant δ.

5.1 The Locally Uniquely Decodable Code

Here we present our code $\mathsf{GGHKR}\colon \{0,1\}^k \to \{0,1\}^{k'}$, that is locally uniquely decodable from $1-\tau$ fraction of agreement by $\mathbf{TC}^0$ circuits for $\tau = \frac{1}{400}$.[23]

Lemma 5.4 (the GGHKR code)**.** *There exists a family of logspace-computable codes*

$$\mathsf{GGHKR}\colon \{0,1\}^k \to \{0,1\}^{k'},$$

where $k' = \mathrm{poly}(k)$, that is locally uniquely decodable from $1-\tau$ fraction of agreement, where $\tau = \frac{1}{400}$, in the following manner.

For every confidence parameter $\delta > 0$, there exists an algorithm $\mathsf{Dec}_{\mathsf{GGHKR}}$ that runs in space $O(\mathrm{loglog}\, k + \mathrm{loglog}(1/\delta))$, takes as input a seed y of length $O(\log(k/\delta))$, and outputs a (deterministic) oracle $\mathbf{TC}^0$ circuit C_y of size

$$s(k,\delta) = \mathrm{polylog}(k) \cdot \widetilde{O}(\log^3(1/\delta)),$$

with the following guarantees.

- *For every $w \in \{0,1\}^{k'}$, and $c = \mathsf{GGHKR}(x)$ for $x \in \{0,1\}^k$, that agrees with w in at least $1-\tau$ fraction of its coordinates,*

$$\Pr_y\big[\forall i \in [k], C_y^w(i) = x_i\big] \geq 1-\delta.$$

- *C_y is non-adaptive. That is, given y and i, there exists an algorithm that runs in space $O(\mathrm{loglog}\, k + \mathrm{loglog}(1/\delta))$ and outputs the coordinates of w to be queried by $C_y(i)$. Consequently, the algorithm $\mathsf{Dec}_{\mathsf{GGHKR}}$, on input y, can output a list $\mathcal{L}(y) \subseteq [k']$ of size $s(k,\delta)\cdot k$ such that C_y only ever queries the received word at locations in $\mathcal{L}(y)$.*

For our approximate uniquely decodable code, we get the following result.

Lemma 5.5 (the $\mathsf{GGHKR}_{\mathsf{LE}}$ code)**.** *There exists a family of logspace-computable codes*

$$\mathsf{GGHKR}_{\mathsf{LE}}\colon \mathbb{F}^k \to \{0,1\}^{k'},$$

parameterized by $d \leq k$, where $k' = \mathrm{poly}(k)$, that is locally approximate list decodable, in the following manner, as long as $|\mathbb{F}|$ is at most exponential in k.

For every confidence parameter $\delta > 0$, there exists an algorithm $\mathsf{Dec}_{\mathsf{GGHKR}_{\mathsf{LE}}}$ that runs in space $O(\log d + \mathrm{loglog}\, k + \mathrm{loglog}\,|\mathbb{F}| + \mathrm{loglog}(1/\delta))$, takes as input a seed y of length $O(\log(kd/\delta))$, and outputs a (determinsitic) oracle $\mathbf{TC}^0$ circuit C_y of size

$$\mathrm{poly}(d, \log k, \log|\mathbb{F}|) + \widetilde{O}(d^2 \log|\mathbb{F}|) \cdot \log^2(1/\delta),$$

with the following guarantees.

[23] The constant $\frac{1}{400}$ is pretty arbitrary, and any small enough constant will do.

- *For every $w \in \{0,1\}^{k'}$, and $c = \mathsf{GGHKR}(x)$ for $x \in \mathbb{F}^k$ that agrees with w in at least $1-\tau$ fraction of its coordinates,*

$$\Pr_y\left[\Pr_{i \leftarrow [k]}\left[C_y^w(i) = x_i\right] \geq 1 - d^{-c}\right] \geq 1 - \delta,$$

  for some universal constant $c \in (0,1)$.

- *C_y makes $Q = \widetilde{O}(d^2 \log|\mathbb{F}|) \cdot \log^2(1/\delta)$ non-adaptive queries. That is, given y and i, there exists an algorithm that runs in the above space and outputs the coordinates of w to be queried by $C_y(x)$. Consequently, the algorithm $\mathsf{Dec}_{\mathsf{GGHKR_{LE}}}$, on input y, can output a list $\mathcal{L}(y) \subseteq [k']$ of size $Q \cdot k$ such that C_y only ever queries the received word at locations in $\mathcal{L}(y)$.*

- *The mapping $\mathsf{GGHKR_{LE}}$ can be computed by logspace-uniform $\mathbf{TC}^0$ circuits, and the decoding map has locality d. More specifically, there exists a logspace-uniform $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(d, \log k)$, that on input $i \in [k']$, returns the d coordinates $q_1, \dots, q_d \in [k]$ of x to be queried, and another logspace-uniform $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(d, \log|\mathbb{F}|)$, that given $x_{q_1}, \dots, x_{q_d}$, outputs $\mathsf{GGHKR_{LE}}(x)_i$.*

We will prove both lemmas in the following subsections.

5.1.1 The Encoding of GGHKR

The encoding follows the one in [DT23], but here we apply another concatenation step to decrease the decoder size, and establish additional stronger properties. Given $x \in \{0,1\}^k$, the encoding $\mathsf{GGHKR}(x) \in \{0,1\}^{k'}$ goes as follows.

Low-degree extension. We encode x into $x^{(1)} \in \mathbb{F}^{|\mathbb{F}|^m}$ via the low-degree extension view of the Reed–Muller code. Specifically, for $|\mathbb{F}| = \log^2 k$, a subset $H \subseteq \mathbb{F}$ of size $\log k$, and $m = \frac{\log k}{\log|H|}$, to encode a string $x \in \{0,1\}^k$ (in fact, any $x \in \mathbb{F}^k$, but the difference won't matter), we first set $p(i) = x_i$ for all $i \in H^m \equiv [k]$ and then extend p to an m-variate polynomial of degree at most $|H|-1$ in each of the m variables. $x^{(1)}$ then comprises all evaluations of p over $\mathbb{F}^m$. Note that $x^{(1)} \in \mathbb{F}^{k_1}$, where $k_1 = k^2$, and denote by $\mathcal{C}_1 \colon \{0,1\}^k \to \mathbb{F}^{k_1}$ the corresponding mapping.

Distance amplification. We employ an [ABN+92]-like distance amplification step that maps $x^{(1)} \in \mathbb{F}^{k_1}$ to $x^{(2)} \in (\mathbb{F}^d)^{k_1}$ by aggregating symbols according to a bipartite expander of degree $d = \mathrm{poly}(\mathbb{F})$. We use expanders with strongly explicit neighborhood functions, computable in $\mathbf{AC}^0$, as in [DT23]. We let $\mathcal{C}_2 \colon \mathbb{F}^{k_1} \to (\mathbb{F}^d)^{k_1}$ denote the distance amplification mapping.

Self-concatenation. Now, let us denote by $\mathcal{C}' = \mathcal{C}'(k) \colon \{0,1\}^k \to \Sigma_k^{k_1}$ the composition $\mathcal{C}_2 \circ \mathcal{C}_1$, parameterized by k, where $\Sigma_k = \mathbb{F}^d$ and $|\Sigma_k| = 2^{\mathrm{polylog}(k)}$. In [DT23], it is shown that $\mathcal{C}'$ can be indeed be computed in space $O(\log k)$. We concatenate $\mathcal{C}'$ with itself twice, instantiated with the appropriate parameters. Namely, let

- $\mathcal{C}'_{(1)} = \mathcal{C}'(k)$ as above.
- $\mathcal{C}'_{(2)} = \mathcal{C}'(\log|\Sigma_k|)$, so $\mathcal{C}'_{(2)}$ has length $\log^2|\Sigma_k|$ and alphabet $\Gamma_k \triangleq \Sigma_{\log|\Sigma_k|}$ of size $2^{\mathrm{polylog}(\log|\Sigma_k|)} = 2^{\mathrm{polyloglog}(k)}$.
- $\mathcal{C}'_{(3)} = \mathcal{C}'(\log|\Gamma_k|)$, so $\mathcal{C}'_{(3)}$ has length $\log^2|\Gamma_k|$ and alphabet $\Lambda_k \triangleq \Sigma_{\log|\Gamma_k|}$ of size $2^{\mathrm{polylog}(\log|\Gamma_k|)} = O(\log k)$.

We concatenate $\mathcal{C}'_{(1)}$ with $\mathcal{C}'_{(2)}$, and concatenate the resulting code with $\mathcal{C}'_{(3)}$, denoting the end result by $\mathcal{C}''$, mapping $x^{(2)}$ to $x^{(3)}$. The alphabet of $\mathcal{C}''$ is clearly Λ_k, and the block length is

$$k_2 = k^2 \cdot \log^2|\Sigma_k| \cdot \log^2|\Gamma_k| = \widetilde{O}(k^2).$$

By composition of space-bounded functions, $\mathcal{C}''(k) \colon \{0,1\}^k \to (\Lambda_k)^{\widetilde{O}(k^2)}$ is computable in space $O(\log k)$.

For our locally-encodable code $\mathsf{GGHKR}_{\mathsf{LE}}$, we skip only the *first* low-degree extension encoding (which indeed cannot be computed locally). Namely, in place of $\mathcal{C}'_{(1)}$, we only do distance amplification. This results in a code $\mathcal{C}''_{\mathsf{LE}}$ that maps $\mathbb{F}^{k_1}$ to $(\Lambda_k)^{\widetilde{O}(k^2)}$. Renaming parameters, and treating the initial alphabet $\mathbb{F}$ and the expander's degree d of the first ABNNR step as parameters, we get that the code $\mathcal{C}''_{\mathsf{LE}} = \mathcal{C}''_{\mathsf{LE}}(k, \mathbb{F}, d)$ maps $\mathbb{F}^k$ to $k \cdot \widetilde{O}(d^2 \log^2|\mathbb{F}|)$ symbols of alphabet of size $2^{\mathrm{polyloglog}(d \log|\mathbb{F}|)}$.

Concatenting with STV. Finally, we map $x^{(3)}$ to the binary $x^{(4)} \in \{0,1\}^{k'}$ by another code concatenation, the STV one [STV01]. Specifically, we encode each symbol in Λ_k by a suitably instantiated Reed–Muller code, concatendated with Hadamard, denoted by $\mathcal{C}'_{(4)} \colon \Lambda_k \to \{0,1\}^{\mathrm{polylog}|\Lambda_k|}$. The length of $\mathcal{C}'_{(4)}$ is already small enough for a naive encoding in linear space. We denote the concatenation of $\mathcal{C}''$ with $\mathcal{C}'_4$ by $\mathcal{C}'''$, having block length $k' = \widetilde{O}(k^2) \cdot \mathrm{polylog}|\Lambda_k| = \widetilde{O}(k^2)$. (Or $k \cdot \widetilde{O}(d^2 \log^2|\mathbb{F}|)$ in the case of $\mathcal{C}'''_{\mathsf{LE}}$.)

We record the above construction in the following two claims.

Claim 5.6 (encoding of GGHKR)**.** *For any positive integer k, the code $\mathsf{GGHKR} \colon \{0,1\}^k \to \{0,1\}^{k'=\widetilde{O}(k^2)}$ above is computable in space $O(\log k)$.*

Claim 5.7 (encoding of $\mathsf{GGHKR}_{\mathsf{LE}}$)**.** *For any positive integers k and $d \le k$, and any alphabet $\mathbb{F}$ of size at most exponential in k, the code*

$$\mathsf{GGHKR}_{\mathsf{LE}} \colon \mathbb{F}^k \to \{0,1\}^{k'=\widetilde{O}(kd^2)}$$

is computable in space $O(\log k)$. Moreover, given $x \in \mathbb{F}^k$ and $i \in [k']$, $\mathsf{GGHKR}_{\mathsf{LE}}(x)_i$ can be computed by making at most d queries to x.

The locality property readily follows from the fact that we only need to query the d neighbors of i in the bipartite expander.

The circuit complexity of $\mathsf{GGHKR}_{\mathsf{LE}}$. In Section 6, we will need a stronger guarantee on the encoding map $\mathsf{GGHKR}_{\mathsf{LE}}$. Not only should it be logspace computable, but in fact encodable by $\mathbf{TC}^0$ circuits that can be generated in small space.

Recall that the first part of the encoding $\mathsf{GGHKR}_{\mathsf{LE}}(x)_i$ is to determine the coordinates of x to be queried.

Claim 5.8 (circuit complexity of $\mathsf{GGHKR}_{\mathsf{LE}}$ – I)**.** *There exists an algorithm that runs in space $O(\mathrm{loglog}\, k + \log d)$, and outputs a $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(d, \log k)$ that on input $i \in [k']$, returns the d coordinates $q_1, \ldots, q_d \in [k]$ to be queried. (That is, for all x, $\mathsf{GGHKR}_{\mathsf{LE}}(x)_i$ only depends on $x_{q_1}, \ldots, x_{q_d}$.)*

Proof. Let $\Gamma \colon [k] \times [d] \to [k]$ be the bipartite expander used in the first step of the encoding, and recall that Γ is strongly explicit, and furthermore it is computable in polynomial-sized $\mathbf{AC}^0$ circuits. Moreover, $\Gamma(u, j)$ is computed by taking a walk of length $O(\log d)$, labeled by j, on an undirected graph (in particular, the Margulis–Gabber–Galil expander), starting from the vertex u. Thus, to iterate over the set $\Gamma^{-1}(v)$, one can simply iterate over $\Gamma(v, j)$ for all j-s.

Given a coordinate $i \in [k']$, determining $q_1, \ldots, q_d$ can be done as follows. Recall that $\mathsf{GGHKR}_{\mathsf{LE}}(x)_i$ is obtained by an alphabet enlargement step $x \mapsto x' \in (\mathbb{F}^d)^k$ according to Γ, and then encoding each element of x' by an inner code. So first, we need to map the input i to the unique $i' \in [k]$ so that $\mathsf{GGHKR}_{\mathsf{LE}}(x)_i$ is part of the encoding of $x'_{i'}$. This can be done by standard arithmetic of integers with $O(\log k)$ bits, and in particular using logspace-uniform $\mathbf{TC}^0$ circuits of size $\mathrm{polylog}(k)$.

Once we computed i', we need to output the set $\Gamma^{-1}(i')$. This can be done by going over all $\Gamma(i', j)$ for $j \in [d]$. Each computation can be implemented by a logspace-uniform $\mathbf{AC}^0$ circuit of size $\mathrm{polylog}(k)$, and so the size bound follows. ∎

Next, once we have $x_{q_1}, \ldots, x_{q_d}$, we want a $\mathbf{TC}^0$ circuit that computes $\mathsf{GGHKR}(x)_i$.

Claim 5.9 (circuit complexity of $\mathsf{GGHKR}_{\mathsf{LE}}$ – II)**.** *There exists an algorithm that runs in space $O(\log d + \mathrm{loglog}\, |\mathbb{F}|)$, and outputs a $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(d, \log |\mathbb{F}|)$ that given the above $x_{q_1}, \ldots, x_{q_d} \in \mathbb{F}$, outputs $\mathsf{GGHKR}_{\mathsf{LE}}(x)_i$.*

Proof. The required complexity property is closed under (a constant number of) compositions and concatenations, so it suffices to argue for each code separately.

- The output of the ABNNR step $\mathcal{C}_2$ is given to us as input.
- Low degree extension and distance amplification, $\mathcal{C}'(M)$, for message lengths $M \leq d \log |\mathbb{F}|$. We already saw that the distance-amplification step can be done by the

appropriate TC^0 circuits (or even in AC^0), so it's left to establish the fact that low-degree extension can be done efficiently enough, and we will use the notation of Section 5.1.1.

Given the subset $H \subseteq \mathbb{F}'$ of size $\log M$, where $|\mathbb{F}'| = \log^2 M$, given a function $x \colon H^m \to \mathbb{F}'$, where $m = \frac{\log M}{\log |H|}$, the unique extension to $f \colon \mathbb{F}'^m \to \mathbb{F}'$ can be computed as

$$f(\alpha) = \sum_{h \in H^m} x(h) \cdot L_h(\alpha),$$

where $L_h(z) = \prod_{\beta \in H \setminus \{h\}} \frac{z-\beta}{h-\beta}$ is the Lagrange polynomial. Using the fact that elementary operations in $\mathbb{F}$ can be done in logspace-uniform TC^0 (see, e.g., [RT92]), we get that each $f(\alpha)$ can be computed by a logspace-uniform TC^0 circuit of size

$$\mathrm{poly}(|H|^m \cdot \log |\mathbb{F}'|) = \mathrm{poly}(d, \log |\mathbb{F}|),$$

as desired.

- The STV encoding for message length $\mathrm{polyloglog}(d \log |\mathbb{F}|)$. This too can be done in constant depth (and size $\leq \mathrm{poly}(d, \log(|\mathbb{F}|))$)w3: The Reed–Muller encoding is the same (the fact that we're using a different regime of parameters only affects the decoding), and the Hadamard encoding can be easily done in AC^0. ■

5.1.2 The Uniform Decoding of GGHKR

The uniform decoding of GGHKR (and its variant $\mathsf{GGHKR}_{\mathsf{LE}}$) is similar to the one in [DT23], but here we make it randomness-efficient, and keep track of our use of randomness, since eventually we aim for a deterministic reconstruction.

Decoding the RM code. Observe that the only place we use randomness for the decoding is the choice of a random line in the Reed–Muller decoding $\mathcal{C}_1$ (the STV code will be decoded by brute force). More formally, we choose a random point in $\mathbb{F}^m$, and the other point needed to describe a line is determined by the location we wish to decode. We aim to (uniquely) decode from very small distance, concretely $\delta_1 = \frac{1}{100|\mathbb{F}|}$.[24] Recall that in local decoding of such RM codes, we only need choose a random line that passes through the desired location and query the rest of the coordinates (or some subset of them). By a simple union-bound, a random line, determined by a random point in $\mathbb{F}^m$, will be good with probability at least $\frac{99}{100}$, in the sense that *all* queried points will be errorless. Then, a simple Lagrange interpolation suffices to recover the desired coordinate.

Instead of choosing a point uniformly at random from $\mathbb{F}^m$ (followed by, perhaps, an error-reduction procedure at the end of the decoding procedure), we use a dedicated seed

[24] Instead of $\frac{1}{100|\mathbb{F}|}$, we can replace $|\mathbb{F}|$ with the precise degree of the univariate restriction, namely $(|H| - 1)m$. However, this difference will be meaningless to us.

to sample a few lines, decode using each line, and take the majority vote. This is the same approach taken in [PRZ23]. For the sampling, we use an $(\varepsilon = \frac{1}{200}, \delta)$ sampler

$$\Gamma_k \colon \{0,1\}^r \times [t] \to \mathbb{F}^m$$

given to us in Theorem 3.11. Thus, $t = O(\log(1/\delta))$ and $r = m \log |\mathbb{F}| + O(\log(1/\delta)) = O(\log(k/\delta))$.

This gives us the following claim.

Claim 5.10. *There exists an algorithm* $\mathsf{Dec}_1 = \mathsf{Dec}_1(k)$ *that gets as input a confidence parameter* $\delta > 0$ *and a seed* $y \in \{0,1\}^{r=O(\log(k/\delta))}$*, runs in deterministic space* $O(\operatorname{loglog} k + \operatorname{loglog}(1/\delta))$*, and outputs a (deterministic) oracle* $\mathbf{TC}^0$ *circuit* C_y *of size* $\operatorname{polylog}(k) \cdot \log(1/\delta)$ *with the following guarantees.*

- *For every* $w \in \mathbb{F}^{k_1 = k^2}$*, and* $c = \mathcal{C}_1(m)$ *for* $m \in \{0,1\}^k$ *that agrees with* w *in at least* $1 - \delta_1$ *fraction of its coordinates,*

$$\Pr_y\left[\forall x \in [k], C_y^w(x) = m_x\right] \geq 1 - k \cdot \delta.$$

- C_y *queries* w *in at most* $O(\log^2 k \cdot \log(1/\delta))$ *locations, non-adaptively. That is, given* y *and* x*, there exists an algorithm that runs in space* $O(\operatorname{loglog} k + \operatorname{loglog}(1/\delta))$ *and outputs the coordinates of* w *to be queried by* C_y *on input* x*.*

  Consequently, by taking the union over all x*-s, the algorithm* Dec_1*, on input* y*, can output a list* $\mathcal{L}(y) \subseteq [k^2]$ *of size* $\widetilde{O}(k) \cdot \log(1/\delta)$ *such that* C_y *only ever queries the received word at locations in* $\mathcal{L}(y)$*.*

Proof. We treat the input x as a point in H^m. Given a seed y to the sampler Γ_k, for any $i \in [t]$, consider the decoding procedure described above of passing a line determined by the points x and $\Gamma(y,i) \in \mathbb{F}^m$, querying w along the line, and interpolating to find x.

More formally, let ℓ_i denote the line $x + a \cdot \Gamma(y,i)$ for $a \in \mathbb{F}$. Let $\alpha_1, \ldots, \alpha_{(|H|-1)m+1}$ be distinct elements in $\mathbb{F}^\star$, and for each j, let $z_{i,j} = (f_w \circ \ell_i)(\alpha_j)$, for f_w being the low-degree extension given by w. The i^{th} guess for x, call it $g(x,i)$, is determined by interpolating to find a degree-$(|H|-1)m$ univariate polynomial h_i such that $h_i(\alpha_j) = z_{i,j}$ for all j and then outputting $h_i(0)$. This procedure, essentially Lagrange interpolation, can be done by an oracle $\mathbf{TC}^0$ circuit of size $\operatorname{poly}(|\mathbb{F}|)$, and we can output the circuit's description in space $O(\log|\mathbb{F}|)$ (see [DT23]).

Dec_1 can then run over all $i \in [t]$, compute $\Gamma(y,i)$, and hard-wire them to the decoding circuit it outputs. Note that given x and $\Gamma(y,i)$, the $|\mathbb{F}|$ oracle queries to w are fixed, and computing $g(x,i)$, as noted above, can be done in $\mathbf{TC}^0$ generated in $O(\log|\mathbb{F}|)$ space. All that is to output a description of a circuit that computes the *majority* of the $g(x,i)$-s. Recalling that Γ is computable in logarithmic space, Dec_1 can be implemented to run in space $O(\log t + \log r + \log|\mathbb{F}|) = O(\operatorname{loglog} k + \operatorname{loglog}(1/\delta))$. The size of the circuit that Dec_1 outputs is $t \cdot \operatorname{poly}(|\mathbb{F}|) = \operatorname{polylog}(k) \cdot \log(1/\delta)$. The correctness readily follows from the properties of the sampler and a simple union-bound. ■

Decoding $\mathcal{C}_2$. The ABNNR step is deterministic, so we can use the uniform decoding result from [DT23]. There, it is shown that there exists an oracle TC^0 circuit of size $\mathrm{polylog}(k)$ generated in space $O(\log\log k)$, that on input a location $i \in [k_1]$, queries the received word $w \in (\mathbb{F}^d)^{k_1}$ in d locations, that depend only on i. The parameters are chosen so that we can (uniquely) locally decode from some constant relative distance $\beta_1 = \beta_1(\tau)$.[25] We record the following claim.

Claim 5.11. *There exists an algorithm* $\mathsf{Dec}_2 = \mathsf{Dec}_2(k, d, |\mathbb{F}|)$ *that runs in deterministic space* $O(\log d + \log\log k + \log\log|\mathbb{F}|)$ *and outputs a (deterministic) oracle* TC^0 *circuit* C *of size*

$$\mathrm{poly}(d, \log|\mathbb{F}|, \log k)$$

with the following guarantees.

- *For every* $w \in (\mathbb{F}^d)^{k_1 = k^2}$, *and* $c = \mathcal{C}_2(m)$ *for* $m \in \mathbb{F}^{k_1}$ *that agrees with* w *in at least* $1 - \beta_1$ *fraction of its coordinates,* $C^w(i) = m_i$ *for every* $i \in [k_1]$.
- C *queries* w *in at most* d *locations, non-adaptively. That is, given* i, *there exists an algorithm that runs in space* $O(\log\log k)$ *and outputs the coordinates of* w *to be queried by* C *on input* i.

We will be using this claim with two parameters settings. The first, with GGHKR, $|\mathbb{F}|$ and d are always $\mathrm{polylog}(k)$, in which case the decoding circuit is of size $\mathrm{polylog}(k)$, and can be generated in space $O(\log\log k)$. The second, with $\mathsf{GGHKR}_{\mathsf{LE}}$, both $\mathbb{F}$ and $d \leq k$ are general parameters. Note, however, that this applies to the first distance amplification step in $\mathsf{GGHKR}_{\mathsf{LE}}$. When we self-concatenate, the ABNNR parameters of GGHKR and $\mathsf{GGHKR}_{\mathsf{LE}}$ are the same.

Decoding $\mathcal{C}'$. Composing $\mathcal{C}_1$ and $\mathcal{C}_2$ into $\mathcal{C}'$, we get a local (unique) decoder for $\mathcal{C}'$, from constant error, in the standard manner: We first apply the local decoder for $\mathcal{C}_1$ to produce the query locations, which are then passed to the local decoder of $\mathcal{C}_2$. The decoder of $\mathcal{C}_2$ retrieves the values in the requested locations and passes them back to the decoder of $\mathcal{C}_1$, that computes the requested location. For a given seed y for the local decoder of $\mathcal{C}_1$, all queries are done in parallel. We obtain the following claim, noting that outputting the circuit that performs the decoding requires only elementary manipulations beyond outputting the decoding circuits for each code.

Claim 5.12. *There exists an algorithm* $\mathsf{Dec}' = \mathsf{Dec}'(k)$ *that gets as input a confidence parameter* $\delta > 0$ *and a seed* $y \in \{0,1\}^{r = O(\log(k/\delta))}$, *runs in deterministic space* $O(\log\log k + \log\log(1/\delta))$, *and outputs a (deterministic) oracle* TC^0 *circuit* C_y *of size*

$$s(k) = \mathrm{polylog}(k) \cdot \log(1/\delta) + d \cdot \mathrm{poly}(d, \log k) = \mathrm{polylog}(k) \cdot \log(1/\delta)$$

with the following guarantees.

[25] The exact choice of β_1 will not affect the parameters, and we will instantiate the code with different choices of β_1 when applying it over different lengths.

- *For every $w \in (\mathbb{F}^d)^{k_1}$, and $c = \mathcal{C}'(m)$ for $m \in \{0,1\}^k$ that agrees with w in at least $1-\beta_1$ fraction of its coordinates,*

$$\Pr_y\left[\forall i \in [k], C_y^w(i) = m_i\right] \geq 1 - k \cdot \delta.$$

- *C_y queries w in at most $d \cdot O(\log^2 k \cdot \log(1/\delta)) = \mathrm{polylog}(k) \cdot \log(1/\delta)$ locations, non-adaptively. That is, given y and i, there exists an algorithm that runs in space $O(\operatorname{loglog} k + \operatorname{loglog}(1/\delta))$ and outputs the coordinates of w to be queries by C_y on input i.*

  Consequently, the algorithm Dec', on input y, can output a list $\mathcal{L}(y) \subseteq [k_1]$ of size $\widetilde{O}(k) \cdot \log(1/\delta)$ such that C_y only ever queries the received word at locations in $\mathcal{L}(y)$.

Decoding the self-concatenation. Recall that we concatenate $\mathcal{C}'(k)$ with $\mathcal{C}'(\log|\Sigma_k|)$, and then with $\mathcal{C}'(\log|\Gamma_k|)$. We need the end result, $\mathcal{C}''$, to be locally decodable from constant relative distance, say $\beta = \sqrt{\tau}$. Towards that end, we set the decoding distance of those three codes accordingly, say $\beta^{1/4} < \frac{1}{2}$. To get a local decoder for $\mathcal{C}''$, we employ standard decoding of concatenated codes, but we need to keep track of the randomness we use, since each instantiation of $\mathcal{C}'$ requires a random seed. In more details, on an input $x \in [k]$,

1. We use a seed $y_1 \in \{0,1\}^{r(k)}$ in order to specify $q_1 = \mathrm{polylog}(k) \cdot \log(1/\delta)$ query locations.
2. For every such query, we use the decoder for $\mathcal{C}'(\log|\Sigma_k|)$ to decode the relevant Σ_k-symbol by going over all $\log|\Sigma_k|$ locations. To do so, we need seeds $y_2^{(1)}, \ldots, y_2^{(\log|\Sigma_k|)}$, each of length $r(\log|\Sigma_k|)$.
3. Each such seed specifies $q_2 = \mathrm{polylog}(\log|\Sigma_k|) \cdot \log(1/\delta)$ query locations.
4. For each such query, we use the decoder for $\mathcal{C}'(\log|\Gamma_k|)$ to decode the relevant Γ_k-symbol by going over all $\log|\Gamma_k|$ locations. To do so, we need seeds $y_3^{(1)}, \ldots, y_3^{(\log|\Gamma_k|)}$, each of length $r(\log|\Gamma_k|)$.
5. Each such seed specifies $q_3 = \mathrm{polylog}(\log|\Gamma_k|)\cdot\log(1/\delta)$ query locations to Λ_k-symbols in our received word w.

We start with some bookkeeping. For any given $i \in [k]$ and a sequence of seeds as above, the number of queries to $w \in (\Lambda_k)^{\widetilde{O}(k^2)}$ is

$$Q = q_1 \cdot \log|\Sigma_k| \cdot q_2 \cdot \log|\Gamma_k| \cdot q_3 = \mathrm{polylog}(k) \cdot \log^3(1/\delta). \tag{1}$$

To save randomness, we use the same seed in each level. The total length of seed needed is thus $r(k) + r(\log|\Sigma_k|) + r(\log|\Gamma_k|)$, which is dominated by $r(k) = O(\log(k/\delta))$. The error probability, by a simple union-bound, is $Q \cdot \delta$, and we multiply this by k if we want to succeed for every i using the same seed.

As all queries are done in parallel, the resulting circuit is a TC^0 one, of size

$$s(k) + q_1 \cdot \log|\Sigma_k| \cdot s(\log|\Sigma_k|) + q_1 \cdot q_2 \cdot \log|\Gamma_k| \cdot s(\log|\Gamma_k|) = \mathrm{polylog}(k) \cdot \log^3(1/\delta). \quad (2)$$

We record the above in the following claim.

Claim 5.13. *There exists an algorithm* $\mathsf{Dec}'' = \mathsf{Dec}''(k)$ *that gets as input a confidence parameter* $\delta > 0$ *and a seed* $y \in \{0,1\}^{r=O(\log(k/\delta))}$*, runs in deterministic space* $O(\mathrm{loglog}\, k + \mathrm{loglog}(1/\delta))$*, and outputs a (deterministic) oracle* TC^0 *circuit* C_y *of size* $\mathrm{polylog}(k) \cdot \log^3(1/\delta)$ *with the following guarantees.*

- *For every* $w \in (\Lambda_k)^{k_2 = \widetilde{O}(k^2)}$*, and* $c = \mathcal{C}''(m)$ *for* $m \in \{0,1\}^k$ *that agrees with* w *in at least* $1 - \sqrt{\tau}$ *fraction of its coordinates,*

$$\Pr_y\left[\forall i \in [k], C_y^w(i) = m_i\right] \geq 1 - kQ \cdot \delta.$$

- C_y *queries* w *in at most* $Q = \mathrm{polylog}(k) \cdot \log^3(1/\delta)$ *locations, non-adaptively. That is, given* y *and* i*, there exists an algorithm that runs in space* $O(\mathrm{loglog}\, k + \mathrm{loglog}(1/\delta))$ *and outputs the coordinates of* w *to be queries by* C_y *on input* i*.*

  Consequently, the algorithm Dec''*, on input* y*, can output a list* $\mathcal{L}(y) \subseteq [k_2]$ *of size* $Q \cdot k$ *such that* C_y *only ever queries the received word at locations in* $\mathcal{L}(y)$*.*

Decoding STV. In our STV encoding, we map Λ_k into a string of length $\mathrm{polylog}\,|\Lambda_k| = \mathrm{polyloglog}(\log k)$. We don't need local decoding here, since the block length is very small and trivial decoding will suffice, namely going over all messages and checking them one by one. Specifically, for all $z \in \Lambda_k$, let D_z be the circuit that has $\mathcal{C}'_{(4)}(z)$ hard-coded, gets oracle access to some w and returns the Hamming distance between w and $\mathcal{C}'_{(4)}(z)$. Each D_z is a (multi-output bit) TC^0 circuit of size $\mathrm{polyloglog}(\log k)$ that can be generated, naively, in this much space (or even in quadruple-log space, see [CT21b]). Now, the decoder simply needs to choose the z for which D_z gives the minimal value. This can be implemented by a (non-adaptive) TC^0 circuit of size $\mathrm{poly}(|\Lambda_k|) = O(\log k)$ that can be generated in space $O(\log|\Lambda_k|) = O(\log\log k)$.

The final decoding. We concatenate $\mathcal{C}''$ with $\mathcal{C}'_{(4)}$. This increaes the number of queries Q by only a multiplicative factor of $\log|\Lambda_k|$, which is negligible. Following the same reasoning as above (but not caring about locality), we get the following claim.

Claim 5.14. *There exists an algorithm* $\mathsf{Dec}_{\mathsf{GGHKR}} = \mathsf{Dec}_{\mathsf{GGHKR}}(k)$ *that gets as input a confidence parameter* $\delta > 0$ *and a seed* $y \in \{0,1\}^{r=O(\log(k/\delta))}$*, runs in deterministic space* $O(\mathrm{loglog}\, k + \mathrm{loglog}(1/\delta))$*, and outputs a (deterministic) oracle* TC^0 *circuit* C_y *of size* $\mathrm{polylog}(k) \cdot \log^3(1/\delta)$ *with the following guarantees.*

- *For every $w \in \{0,1\}^{k'}$, and $c = \mathsf{GGHKR}(m)$ for $m \in \{0,1\}^k$ that agrees with w in at least $1-\tau$ fraction of its coordinates,*

$$\Pr_y\left[\forall i \in [k], C_y^w(i) = m_i\right] \geq 1 - kQ \cdot \delta.$$

- *C_y queries w in at most $Q = \mathrm{polylog}(k) \cdot \log^3(1/\delta)$ locations, non-adaptively. That is, given y and i, there exists an algorithm that runs in space $O(\mathrm{loglog}\, k + \mathrm{loglog}(1/\delta))$ and outputs the coordinates of w to be queries by C_y on input i.*

  Consequently, the algorithm $\mathsf{Dec}_{\mathsf{GGHKR}}$, on input y, can output a list $\mathcal{L}(y) \subseteq [k']$ of size $Q \cdot k$ such that C_y only ever queries the received word at locations in $\mathcal{L}(y)$.

Choosing $\bar{\delta}$ such that $kQ \cdot \bar{\delta} = \delta$, recalling that $|y| = O(\log(k/\bar{\delta}))$, and considering the uniform encoding of Claim 5.6, we thereby proved Lemma 5.4.

The final decoding – the local encoding variant. Here, recall that we dispense with the first RM encoding of $\mathcal{C}'(k)$, but keep it in $\mathcal{C}'(\log|\Sigma_k|)$ and $\mathcal{C}'(\log|\Lambda_k|)$. We then still concatenate with STV. Thus, the decoding is the same, only without the top RM decoding, so it is left to just keep track of parameters, leaving the parameters $\mathbb{F}$ and d unset. However, note that $\mathsf{GGHKR}_{\mathsf{LE}}$ is now no longer locally decodable, but only locally approximately decodable.

Let δ_{s} be the confidence parameter of the sampler used in the distance amplification step. We use balanced bipartite expanders, so $\delta_{\mathsf{s}} = d^{-\Omega(1)}$ (see, e.g., [GGH+07]), and we set the parameters so that the accuracy parameter is constant ($\frac{1}{2} - \tau$ suffices).

Claim 5.15. *There exists an algorithm $\mathsf{Dec}_{\mathsf{GGHKR}_{\mathsf{LE}}} = \mathsf{Dec}_{\mathsf{GGHKR}_{\mathsf{LE}}}(k, d, \mathbb{F})$ that gets as input a confidence parameter $\delta > 0$ and a seed $y \in \{0,1\}^r$ for $r = O(\log \frac{d \log|\mathbb{F}|}{\delta})$, runs in deterministic space $O(\log d + \mathrm{loglog}\, k + \mathrm{loglog}\,|\mathbb{F}| + \mathrm{loglog}(1/\delta))$, and outputs a (deterministic) oracle $\mathbf{TC}^0$ circuit C_y of size*

$$\mathrm{poly}(d, \log k, \log|\mathbb{F}|) + \widetilde{O}(d^2 \log|\mathbb{F}|) \cdot \log^2(1/\delta)$$

with the following guarantees.

- *For every $w \in \{0,1\}^{k'}$, and $c = \mathsf{GGHKR}_{\mathsf{LE}}(m)$ for $m \in \mathbb{F}^k$ that agrees with w in at least $1-\tau$ fraction of its coordinates,*

$$\Pr_y\left[\Pr_{i \leftarrow [k]}[C_y^w(i) = m_i] \geq 1 - \delta_{\mathsf{s}}\right] \geq 1 - kQ \cdot \delta.$$

- *C_y queries w in at most $Q = \widetilde{O}(d^2 \log|\mathbb{F}|) \cdot \log^2(1/\delta)$ locations, non adaptively. That is, given y and i, there exists an algorithm that runs in space $O(\log d + \mathrm{loglog}\, k + \mathrm{loglog}\,|\mathbb{F}|)$ and outputs the coordinates of w to be queried by C_y on input i.*

  Consequently, the algorithm $\mathsf{Dec}_{\mathsf{GGHKR}_{\mathsf{LE}}}$, on input y, can output a list $\mathcal{L}(y) \subseteq [k']$ of size $Q \cdot k$ such that C_y only ever queries the received word at locations in $\mathcal{L}(y)$.

The bound on r, using the above notation, is obtained by $r(\log|\Sigma_k|) + r(\log|\Gamma_k|)$, recalling that $\log|\Sigma_k| = d\log|\mathbb{F}|$ and $\log|\Gamma_k| = \mathrm{polylog}(\log|\Sigma_k|)$. To bound the number of queries, we replace q_1 by d in Equation (1). To obtain the bound on the size, we further replace the $s(k)$ term in Equation (2) with the ABNNR decoding size. Finally, the space bound can be inferred by noticing that throughout, it is logarithmic in the circuit's size.

The above lemma, together with the encoding results, imply Lemma 5.5.

5.2 The IW and Had Codes

We now recall the [IW97] and Hadamard codes. The results essentially follow from prior work, but we call attention to two areas where our presentation is nonstandard: We verify that the encoder and decoder can be implemented by logspace uniform $\mathbf{TC}^0$ circuits (following the approach of [CTW23]), and moreover that the decoder uses only logarithmically many random bits (following the approach of [PRZ23]).

Theorem 5.16 ([GL89]). *For every $k \in \mathbb{N}$, the Hadamard code $\mathsf{Had}\colon \{0,1\}^k \to \{0,1\}^{2^k}$ satisfies the following.*

1. ***Uniform Local Encoding.*** *There is a space $O(\log k)$ algorithm that outputs a size $O(k)$ oracle $\mathbf{TC}^0$ circuit C such that $C^f(i) = \mathsf{Had}(f)_i$.*
2. ***Deterministic Decoding.*** *There is a space $O(\log(k/\varepsilon) + \mathrm{loglog}(1/\delta))$ algorithm $\mathsf{Dec}_{\mathsf{Had}}$ that, given $\varepsilon > 0$ and $\delta > 0$ and a seed $y \in \{0,1\}^{O(k+\log(1/\delta))}$, outputs deterministic oracle circuits $C_{y,1}, \dots, C_{y,L}$ with $L = O(k\log(1/\delta)/\varepsilon^2)$, with the following guarantees.*
   - *$C_{y,i}$ is an oracle $\mathbf{TC}^0$ circuit of size $S = \mathrm{poly}(k/\varepsilon)$ with one majority gate, that makes non-adaptive oracle queries.*
   - *For every $w \in \{0,1\}^{2^k}$ and $\bar{f} = \mathsf{Had}(f)$, with agreement at least $1/2 + \varepsilon$, it holds that*
   $$\Pr_y[\exists i \in [L], \forall x \in [k], C^w_{y,i}(x) = f_x] \geq 1 - \delta.$$

Item 1 is immediate from the definition of the Hadamard code. The proof of Item 2 closely follows that of [PRZ23], with the exception that we require the small-bias space be strongly explicit. To prove Item 2, we first show that we can list decode with constant advantage.

Claim 5.17. *There is a space $O(\log(k/\varepsilon))$ algorithm $\mathsf{Dec}_{\mathsf{Had}}$ that, given $\varepsilon > 0$ and a seed $y \in \{0,1\}^{O(k)}$, outputs deterministic oracle circuits $C_{y,1}, \dots, C_{y,L}$ with $L = O(k/\varepsilon^2)$, with the following guarantees.*

- *$C_{y,i}$ is an oracle $\mathbf{TC}^0$ circuit of size $S = \mathrm{poly}(k/\varepsilon)$ with one majority gate, that makes non-adaptive oracle queries.*
- *For every $w \in \{0,1\}^{2^k}$ and $\bar{f} = \mathsf{Had}(f)$, with agreement at least $1/2 + \varepsilon$, it holds that*
$$\Pr_y[\exists i \in [L], \forall x \in [k], C^w_{y,i}(x) = f_x] \geq 2/3.$$

Proof. Let $\ell = \lceil \log(ck/\varepsilon^2 + 1) \rceil$ for a sufficiently large constant c to be chosen later, and define $L = 2^\ell$. Let $\mathsf{Bias}\colon \{0,1\}^{t=O(k)} \to \{0,1\}^{k\cdot\ell}$ be the small bias generator of Proposition 3.14 with output length $k \cdot \ell$ and error $\varepsilon = 2^{-4k}$, and note that the output can be computed in space $O(\log k)$. Then, $\mathsf{Dec}_{\mathsf{Had}}$ operates as follows. Let

$$(v_1, \dots, v_\ell) \triangleq \mathsf{Bias}(y), \quad (b_1, \dots, b_\ell) \triangleq \langle i \rangle,$$

and for $J \subseteq [\ell]$ let $b^J = \oplus_{i \in J} b_i$ and $v^J = \oplus_{i \in J} v_i$, and let e_i for $i \in [k]$ be the i^{th} standard basis vector. Then, $\mathsf{Dec}_{\mathsf{Had}}$ outputs the circuit

$$C^w_{y,i}(x) = \mathsf{MAJ}_{J \subseteq [\ell] : J \neq \emptyset}(b^J \oplus w_{v^J \oplus e_x}).$$

By [PRZ23, Lemma 4.13], we have that the distribution of $(v^J, v^{J'})$ is 2^{-2k}-close to U_{2k} in ℓ_1 distance for every $J \neq J'$.

Now fix w and $\bar{f} = \mathsf{Had}(f)$ with agreement at least $1/2 + \varepsilon$. For every $x \in [k]$, let

$$S_x = \{v : w_{v \oplus e_x} = \bar{f}_{v \oplus e_x} \iff w_{v \oplus e_x} = \langle f, v \oplus e_i \rangle\}.$$

Furthermore, observe that $|S_x| \geq (1 + \varepsilon)2^k$ for every x, which follows from $U_k \oplus e_x$ being uniform over $\{0,1\}^k$ for every k.

Lemma 5.18. *We have that for every $x \in [k]$,*

$$\Pr_y\left[|\{J : v^J \in S_x\}| \geq 2^{\ell-1} + 1\right] \geq 1 - \frac{1}{100k}.$$

This follows using the exact same proof as [PRZ23, Claim 4.18] (the only difference is that in [PRZ23] the constant c was chosen to be 128 and the error bound was $1/2k$, whereas we choose a sufficiently large $c > 128$ and deduce an error bound of $1/100k$). Thus by Lemma 5.18, with probability 0.99 over y, we have that for every x,

$$|\{J : v^J \in S_x\}| \geq 2^{\ell-1} + 1.$$

We claim that for every y with this property, the circuit $C_{y,i}$, where $\langle i \rangle = (b_1, \dots, b_\ell)$ is such that $b_j = \langle f, v_j \rangle$, satisfies the decoding property. Fixing an arbitrary x, we have that for every J where $v^J \in S_x$ (which occurs for a majority of the J),

$$b^J \oplus w_{v^J \oplus e_x} = \langle f, v^J \rangle \oplus \langle f, v^J \oplus e_x \rangle = \langle f, e_x \rangle,$$

and hence the circuit is correct on input x for every x. Each circuit $C_{y,i}$ is clearly of size $S = O(k \cdot L)$ as claimed, and makes only non-adaptive oracle queries. ∎

Proof of Item 2. Let $\mathsf{Samp} : \{0,1\}^{s+O(\log(1/\delta))} \times [q] \to \{0,1\}^s$ be the sampler of Theorem 3.11 set with accuracy $\varepsilon = 0.1$ and confidence δ (so $q = O(\log(1/\delta))$), and note that each output bit of the sampler is computable in space $O(\log s + \log\log(1/\delta)) = O(\log(k) + \log\log(1/\delta))$. $\mathsf{Dec}_{\mathsf{Had}}$ takes in $y \in \{0,1\}^{s+O(\log(1/\delta))}$ and lets

$$(y_1, \dots, y_q) \triangleq (\mathsf{Samp}(y, 1), \dots, \mathsf{Samp}(y, q)).$$

Then, let C^i be the list of circuits produced by Claim 5.17 with a random seed y_i, and let the final output be $\cup_{i\in[q]} C^i$. Note that we can compose the output of the sampler with the procedure from Claim 5.17 in overall space $O(\log(k/\varepsilon) + \log\log(1/\delta))$, by Proposition 3.5. Finally, it is clear that we fulfill the decoding promise with probability at least $1-\delta$. ■

We now state the IW code with deterministic decoding.

Lemma 5.19 ([IW97]). *There exists a constant $c_{\mathsf{IW}} > 1$ such that for any two constants $\tau_{\mathsf{IW}}, \gamma_{\mathsf{IW}} > 0$ and any $\varepsilon_{\mathsf{IW}} > 0$, the following holds. There exists a code $\mathsf{IW}\colon \{0,1\}^N \to (\{0,1\}^t)^{N'}$ with*

$$t = (c_{\mathsf{IW}}/\tau_{\mathsf{IW}}{}^2) \cdot \log(1/\varepsilon_{\mathsf{IW}}), \quad N' = (N/\varepsilon_{\mathsf{IW}})^{c_{\mathsf{IW}}(1/\gamma_{\mathsf{IW}} + 1/\tau_{\mathsf{IW}}{}^2)},$$

with the following properties:

- ***Uniform Encoding.*** *There is a space $O(\log N)$ algorithm $\mathsf{Enc}_{\mathsf{IW}}$ that outputs an oracle $\mathbf{TC}^0$ circuit C of size $(\log(N)\cdot t)^{c_{\mathsf{IW}}}$, such that $C^f(z,i) = \mathsf{IW}(f)_{(z,i)}$ for every $z \in [N']$ and $i \in [t]$.*
- ***Approximate Local List Decoding.*** *There exists a space $O(\log(N/\delta))$ algorithm $\mathsf{Dec}_{\mathsf{IW}}$ that gets as input $\delta_{\mathsf{IW}} > 0$, a seed $z \in \{0,1\}^{O(\log(N/\delta))}$ and oracle access to a word $f \in \{0,1\}^N$. $\mathsf{Dec}_{\mathsf{IW}}$ makes at most $S = \mathrm{polylog}(1/\delta_{\mathsf{IW}})\cdot N^{\gamma_{\mathsf{IW}}}/\varepsilon_{\mathsf{IW}}{}^{c_{\mathsf{IW}}}$ non-adaptive oracle queries to f and outputs a circuit C_z satisfying the following:*
  - *C_z is a deterministic oracle $\mathbf{TC}^0$ circuit of size S that has one majority gate of fan-in at most $Q = (\log(N)\log(1/\delta_{\mathsf{IW}})/\varepsilon_{\mathsf{IW}})^{c_{\mathsf{IW}}}$, and makes at most Q non-adaptive oracle queries.*
  - *For every $w \in (\Sigma)^{N'}$ with agreement at least ε with $\bar{f} = \mathsf{IW}(f)$ (over the coordinates $i \in [N']$, viewing each coordinate as a symbol in $\Sigma = \{0,1\}^t$), with probability $1-\delta_{\mathsf{IW}}$ over z,*

  $$\Pr_x[C_z^w(x) = f_x] \geq 1 - \tau_{\mathsf{IW}}.$$

For clarity of presentation, for the remainder of the section set $\tau = \tau_{\mathsf{IW}}$, $\gamma = \gamma_{\mathsf{IW}}$, $\varepsilon = \varepsilon_{\mathsf{IW}}$, $\delta = \delta_{\mathsf{IW}}$, and let $n = \log N$. Moreover, as if $\varepsilon_{\mathsf{IW}} < N^{-1} < N^{1/c_{\mathsf{NW}}}$ the statement is trivial, we assume that $\varepsilon > 1/N$ for the remainder of the proof. Set $t = c(1/\tau^2)\log(1/\varepsilon)$ for a sufficiently large constant $c > 1$ to be determined later.

The Construction. We initialize the code using the following two ingredients:

- The sampler $\mathsf{Samp}\colon \{0,1\}^{m_1} \times [t] \to [N]$ of Theorem 3.11 with accuracy $\tau/2$ and confidence $\varepsilon/8$. Note that this implies $m_1 = \log N + O(t)$.
- A design $\mathsf{Des}\colon \{0,1\}^{m_2} \times [t] \to [N]$ with $\alpha = \gamma/2$. With these parameters, $m_2 = O(\frac{1}{\gamma}\log N)$ and t can be as large as $N^{\gamma/c}$ for a universal constant c.

Now let $N' = 2^{m_1} \cdot 2^{m_2}$ so that $\log N' = m_1 + m_2 = O(n/\gamma + \log(1/\varepsilon)/\tau^2)$. For $\bar{z} = (z_1, z_2) \in \{0,1\}^{m_1} \times \{0,1\}^{m_2}$ and $i \in [t]$, let

$$\overline{\mathsf{Loc}}(\bar{z}, i) = \mathsf{Samp}(z_1, i) \oplus \mathsf{Des}(z_2, i) \in [N].$$

Then, define $\bar{f} = \mathsf{IW}(f)$, where for every $\bar{z}$,

$$\bar{f}_{\bar{z}} = (f_{\overline{\mathsf{Loc}}(\bar{z},1)}, \ldots, f_{\overline{\mathsf{Loc}}(\bar{z},t)}).$$

The Encoding. To show that the encoding satisfies the desired properties, we verify the construction of [CTW23] is logspace-uniform (whereas their result is stated as **P**-uniform). To prove this, it suffices to prove that we can output a TC^0 circuit that computes $\overline{\mathsf{Loc}}$ in space $O(\log N)$.

Claim 5.20. *There is a space $O(\log N)$ algorithm that outputs a TC^0 circuit that computes $\overline{\mathsf{Loc}}$ of size $\mathrm{poly}(t \cdot \log N)$.*

Proof. By Theorem 3.11, there is a space $O(\log m_1) = O(\mathrm{loglog}(N/\varepsilon))$ algorithm that outputs an $\mathsf{AC}^0[\oplus]$ circuit of size $\mathrm{poly}(m_1) = \mathrm{polylog}(N)$ that computes $\mathsf{Samp}(\cdot,\cdot)$. For the design, we recall by Theorem 3.16 that there exist designs computable in space $O(\log N)$ with the desired parameters. We compute this design and hardwire it into the circuit (which is of size at most tn), so that given z_2 and i, the circuit can compute $\mathsf{Des}(z_2, i)$. Then, we can easily XOR the two values together, and do this in parallel for every $i \in [t]$, resulting in a TC^0 circuit of size $\mathrm{poly}(t \cdot \log N)$. ■

The Decoding. The decoding follows from the approach of [PRZ23], where we sample many probabilistic circuits using a sampler, and have the decoding circuit take the majority over their output. Note that if $\delta < 2^{-N}$ the statement is trivial, so we assume $\delta \geq 2^{-N}$. We recall the probabilistic circuit from [DT23]:

Lemma 5.21 ([DT23], Lemma A.2). *There exists a space $O(\log N)$ algorithm that, given a seed $y \in \{0,1\}^{\ell = \log(N') - \log(N) + \log(t) + t + 1}$, and oracle access to $f \in \{0,1\}^N$, acts as follows. The algorithm makes $(t-1) \cdot N^{(\gamma/2)}$ queries to f, and prints a circuit F_y such that the following holds.*

- *For every $w \in \Sigma^{N'}$ and $\bar{f} = \mathsf{IW}(f)$ with agreement at least ε (over the coordinates $i \in [N']$, viewing each coordinate as a symbol in $\{0,1\}^t$), for at least $1 - \tau/2$ fraction of the inputs $x \in \{0,1\}^N$, we have $\Pr_y[F_y^w(x) = f_x] \geq 1/2 + \varepsilon/64$.*
- *F_y is an oracle AC^0 circuit of size $O(tN^{\gamma/2})$ that makes a single oracle query.*

It is then easy to use the approach of [PRZ23] to obtain the claim about decoding.

Proof of Lemma 5.19. Let $\mathsf{Samp}\colon \{0,1\}^m \times [Q] \to \{0,1\}^\ell$ be the strong sampler of Theorem 3.12 with accuracy $\varepsilon/128$ and confidence $\delta\tau/2$ (note that this is not the same sampler as in the encoding step). Note that with this choice of parameters we have $Q =$

$\text{poly}(\log(1/\tau\delta)/\varepsilon)$ and $m = \ell + O(\log(1/\varepsilon\tau\delta)) = O(\log(N/\delta))$, and the sampler can be computed in space $O(\log(N/\delta))$. Then, $\mathsf{Dec}_{\mathsf{IW}}$ operates as follows. Letting

$$(y_1, \ldots, y_Q) \leftarrow \mathsf{Samp}(z, \cdot),$$

$\mathsf{Dec}_{\mathsf{IW}}$ outputs the circuit

$$C_z^w(x) = \mathsf{MAJ}_{i \in [Q]}(F_{y_i}^w(x)),$$

where the F_{y_i}-s are constructed using the algorithm of Lemma 5.21. The circuit is of size

$$S = Q \cdot |F_y| = \text{poly}(\log(1/\tau\delta)/\varepsilon) \cdot O(tN^{(\gamma/2)}) = \text{polylog}(1/\delta) \cdot N^{\gamma}/\varepsilon^{c},$$

has a single majority gate of fan-in Q, and makes at most Q non-adaptive oracle queries, as claimed. Moreover, $\mathsf{Dec}_{\mathsf{IW}}$ makes at most $Q \cdot (t-1) \cdot N^{(\gamma/2)} \leq S$ oracle queries in total.

We now argue the second property holds. Fix an arbitrary $w \in \Sigma^{n'}$ and f where w has agreement at least ε with $\bar{f} = \mathsf{IW}(f)$. By Lemma 5.21, there is a set $G \subseteq \{0,1\}^n$ of density at least $1 - \tau/2$ such that for every $x \in G$,

$$\Pr_y[F_y^w(x) = f_x] \geq \frac{1}{2} + \frac{\varepsilon}{64},$$

and thus

$$\Pr_z[C_z^w(x) = f_x] \geq 1 - \frac{\delta\tau}{2}.$$

Call z *good* if C_z^w is incorrect on at most a $\tau/2$ fraction of x in G. By an averaging argument, z must be good with probability at least $1 - \delta$. For every good z, we satisfy the decoding property. ■

Lemma 5.19 for the code $\mathcal{C}_{\mathsf{LE}}$. For the *approximate* locally decodable code, that does not invoke an initial step of low-degree extension decoding, we will need of a variant of Lemma 5.19 in which it is not $\mathsf{Dec}_{\mathsf{IW}}$ that makes the queries to f, but rather C_y itself. This is easy to achieve: $\mathsf{Dec}_{\mathsf{IW}}$ will still make the sampler queries and compute the design, all of which be hard-coded into F_{y_i}. Looking at [DT23, Lemma A.2], we see that we need to query f in locations specified by $\overline{\mathsf{Loc}}$. Following Claim 5.20, this can be done in size $\text{poly}(t \cdot \log N)$ by $\mathbf{TC}^0$ circuits generated in space $O(\log N)$. Thus, each $F_{z_i}^{f,w}$, beyond the queries to w, will first retrieve the required coordinate of f. This adds at most $O(t \cdot N^{\gamma/2}) < S$ queries, we can still take the size bound to be S (incurring only $\text{poly}(t \cdot \log N)$ in size), and finally each $F_{z_i}^{f,w}$ can still be generated in the allotted space $O(\log N)$.

The Composed Code. We collect together the statement of the composed code $\mathsf{IW} \circ \mathsf{Had}$.

Lemma 5.22. *There exists a constant $c > 1$ such that for any constants $\tau, \gamma > 0$, the following holds. There exists a code*

$$\mathcal{C} \colon \{0,1\}^N \to \{0,1\}^{\hat{N}} \text{ with } \hat{N} = (N/\varepsilon)^{c(1/\gamma + 1/\tau^2)},$$

for any $\varepsilon > 0$, with the following properties.

- ***Uniform Encoding.*** *There is a space $O(\log(N/\varepsilon))$ algorithm $\mathsf{Enc}_{\mathcal{C}}$ that outputs an oracle TC^0 circuit C of size $(\log(N/\varepsilon))^{c(1/\gamma+1/\tau^2)}$, such that $C^f(i) = \mathcal{C}(f)_{(i)}$.*
- ***Approximate Local Decoding.*** *There exists a space $O(\log(N/\varepsilon\delta))$ algorithm $\mathsf{Dec}_{\mathcal{C}}$ that gets as input a seed $y \in \{0,1\}^{O(\log(N)+\log(1/\delta))}$, $i \in [L]$ for $L = \widetilde{O}(\log(1/\delta)/\varepsilon^2)$, and oracle access to a word $f \in \{0,1\}^N$. $\mathsf{Dec}^f_{\mathcal{C}}(y,i)$ makes at most $S = N^\gamma \cdot (\log(1/\delta)/\varepsilon)^c$ non-adaptive oracle queries to f and outputs a circuit $C_{y,i}$ satisfying the following:*
  - *$C_{y,i}$ is a deterministic TC^0 oracle circuit of size S with $Q = (\log(N)\cdot\log(1/\delta)/\varepsilon)^c$ majority gates of fan-in at most Q, and makes at most Q non-adaptive oracle queries.*
  - *For every $w \in \{0,1\}^{N'}$ and $\bar{f} = \mathcal{C}(f)$ with agreement at least $1/2+\varepsilon$, with probability $1-\delta$ over y, there exists i such that*
  $$\Pr_x[C^w_{y,i}(x) = f_x] \geq 1-\tau.$$

  Moreover, the algorithm $\mathsf{Dec}_{\mathcal{C}}$ on input y can output a list $\mathcal{L}(y) \subseteq [N]$ of size S such that $\mathsf{Dec}^f_{\mathcal{C}}(y,i)$ only ever queries f at locations in $\mathcal{L}(y)$.

Proof. Let
$$\mathsf{IW}\colon \{0,1\}^N \to (\{0,1\}^k)^{N'}, \quad \mathsf{Had}\colon \{0,1\}^k \to \{0,1\}^{2^k}$$
be the codes of Lemma 5.19 with
$$\tau_{\mathsf{IW}} = \tau, \qquad \gamma_{\mathsf{IW}} = \gamma, \qquad \varepsilon_{\mathsf{IW}} = \Theta\left(\frac{1}{\varepsilon^4\log^2(1/\delta)}\right)$$
and Theorem 5.16 respectively. Note that $k = (c/\tau^2)\cdot\log(1/\varepsilon_{\mathsf{IW}})$ and $N' = (N/\varepsilon_{\mathsf{IW}})^{c(1/\gamma+1/\tau^2)}$. Let the composed code be $\mathcal{C} = \mathsf{Had}\circ\mathsf{IW}$, so
$$\hat{N} = 2^k\cdot N' = (N'/\varepsilon_{\mathsf{IW}})^{2c(1/\gamma+1/\tau^2)}.$$
For $j \in [N']$, let I_j be the bits in the final code corresponding to the Hadamard encoding of the j^{th} symbol of IW.

Encoding. The encoding statement is immediate given the choices of parameters.

Constructing the Decoder. We now construct the decoder. Let $\mathsf{Dec}_{\mathsf{Had}}$ be the decoder of Theorem 5.16 with
$$\varepsilon_{\mathsf{Had}} = \varepsilon/2, \qquad \delta_{\mathsf{Had}} = \delta/4,$$
and note that $\mathsf{Dec}_{\mathsf{Had}}$ runs in space $O(\log(k/\varepsilon) + \operatorname{loglog}(1/\delta))$. Let $\mathsf{Dec}_{\mathsf{IW}}$ be the decoder of Lemma 5.19 with $\delta_{\mathsf{IW}} = \delta/4$ and note that $\mathsf{Dec}_{\mathsf{IW}}$ runs in space $O(\log(N/\delta))$ and we have list size
$$L = O(k\log(1/\delta)/\varepsilon^2) = O(\log(1/\varepsilon\delta)\operatorname{loglog}(1/\delta)/\varepsilon^2) = \widetilde{O}(\log(1/\delta)/\varepsilon^2).$$

The final decoder takes in $y = (y_1, y_2)$ and i, where

$$y_1 \in \{0,1\}^{O(\log(N/\delta))}, \quad y_2 \in \{0,1\}^{O(k+\log(1/\delta))}, \quad i \in [L].$$

The decoder $\mathsf{Dec}_{\mathcal{C}}^{f}(y)$ instantiates $\mathsf{Dec}_{\mathsf{IW}}^{f}(y_1)$. Moreover, define

$$(H_1, \ldots, H_L) \leftarrow \mathsf{Dec}_{\mathsf{Had}}(y_2),$$

where we slightly abuse notation and let $H_j \colon \emptyset \to \{0,1\}^k$ be an oracle circuit that takes no input and returns the entire k-bit decoded value. Now, whenever $\mathsf{Dec}_{\mathsf{IW}}^{f}(y)$ prints an oracle gate for IW, instead print H_i, where if the oracle gate receives index j the circuit gives H_i the bits corresponding to the Hadamard encoding of symbol j.

Success Probability of the Decoder. It now suffices to argue the decoder satisfies the desired properties. Fix $w \in \{0,1\}^{N'}$ and $\bar{f} = \mathcal{C}(f)$ with agreement at least $1/2 + \varepsilon$ For every $j \in [N']$, let I_j be the bits of $\bar{f}$ corresponding to the Hadamard encoding of $\mathsf{IW}(f)_j$. Let

$$G = \left\{ j : w_{I_j} \text{ and } \bar{f}_{I_j} \text{ have agreement at least } 1/2 + \varepsilon \right\}.$$

By an averaging argument, G has density at least $\varepsilon/2$. Moreover, for every $j \in G$, by Theorem 5.16,

$$\Pr_{y_2}\left[\exists i,\ H_i^{w_{I_j}} = \mathsf{IW}(f)_j\right] \geq 1 - \frac{\delta}{4},$$

so the probability that y_2 decodes at least $1/2$ of $j \in G$ is at least $1 - \delta/2$.

Call such a y_2 *good*, so for every good y_2 there exists $i \in [L]$ such that at least an $(\varepsilon/4L)$ fraction of symbols of IW are decoded correctly. Recall that $\varepsilon_{\mathsf{IW}} = \Theta(\varepsilon^{-4}/\log^2(1/\delta))$, so $k = O(\log(\log(1/\delta)/\varepsilon_{\mathsf{IW}})/\tau^2)$ (and by choosing appropriate constants, we have that $\varepsilon_{\mathsf{IW}} \leq (\varepsilon/4L)$). Thus, for every good y_2 there is some fixed i such that $H_i^{w_{I_j}}$ is correct on at least an $(\varepsilon/200L) \geq \varepsilon_{\mathsf{IW}}$ fraction of symbols. In this case, the circuit printed by $\mathsf{Dec}_{\mathsf{IW}}(y_1)$ has the input promise for the decoder satisfied, so with probability at least $\delta/2$ over y_1, we print a circuit that decodes w to agreement at least $1-\tau$. Thus, the total failure probability is at most δ, as desired.

Complexity of the Circuit. Finally, the decoder makes at most

$$S' = \mathrm{polylog}(1/\delta) \cdot N^{\gamma}/{\varepsilon_{\mathsf{IW}}}^{c_{\mathsf{IW}}} = N^{\gamma} \cdot (\log(1/\delta)/\varepsilon)^{c}$$

non-adaptive oracle queries by Lemma 5.19, and outputs a circuit of size $S' \cdot \mathrm{poly}(k/\varepsilon) = N^{\gamma} \cdot (\log(1/\delta)/\varepsilon)^{c} = S$ as claimed. Moreover, this circuit contains a top majority gate of fan-in $Q = (\log(N)\log(1/\delta)/\varepsilon_{\mathsf{IW}})^{c_{\mathsf{IW}}} = (\log(N)\log(1/\delta)/\varepsilon)^{c}$, and further majority gates of fan in at most $\mathrm{poly}(k/\varepsilon) \leq Q$. ■

Lemma 5.22 for the code $\mathcal{C}_{\mathsf{LE}}$. We carry over our alternative decoding of IW, wherein $\mathsf{Dec}_{\mathsf{IW}}$ does not make oracle queries to f and they are deferred to the generated circuits. Here too, we can make $C_{y,i}$ make the (at most) S queries to f themselves.

5.3 Putting Everything Together

We continue using the notation introduced in this section. The code $\mathcal{C}\colon \{0,1\}^k \to \{0,1\}^n$ is the concatenation of $\mathsf{IW} \circ \mathsf{GGHKR}$ with the Hadamard code. Namely, given $x \in \{0,1\}^k$, write $y = \mathsf{IW}(\mathsf{GGHKR}(x))$, and encode each symbol of y with Hadamard. That is,

$$\mathcal{C}(x) = \mathsf{Had}(y_1) \circ \ldots \circ \mathsf{Had}(y_{k''}) \in \{0,1\}^n.$$

For the local encoding variant, $\mathcal{C}_{\mathsf{LE}}$, we replace GGHKR with $\mathsf{GGHKR}_{\mathsf{LE}}$.

Let $\mathcal{C}_{\mathsf{c}}\colon \{0,1\}^{k'} \to \{0,1\}^n$ be the concatenation of IW with the Hadmard code above. We use δ_1 and δ_2 for the confidence parameters of GGHKR and $\mathcal{C}_{\mathsf{c}}$, respectively. Recall that τ is the (relative) unique decoding radius of GGHKR. We also let $\gamma > 0$ be as in Lemma 5.22. Note that the output length of $\mathcal{C}$ is

$$n = (k/\varepsilon)^{c\cdot\left(\gamma^{-1}+\tau^{-2}\right)}$$

for some universal constant $c > 0$, and similarly for $\mathcal{C}_{\mathsf{LE}}$. It is left to establish the local list decoding of the composition $\mathcal{C} = \mathcal{C}_{\mathsf{c}} \circ \mathsf{GGHKR}\colon \{0,1\}^k \to \{0,1\}^n$.

Local list decoding of $\mathcal{C}$. Given a message length k and a confidence parameter $\delta > 0$, set $\delta_1 = \delta_2 = \delta/2$. We instantiate the code $\mathcal{C}_{\mathsf{c}}$ from Lemma 5.22 with $N = k'$, where $k' = \mathrm{poly}(k)$ is the block length of GGHKR from Lemma 5.4 on inputs of length k.

We apply standard local list decoding of composed codes (as also used above in Section 5.1). Let $d_1 = d_1(\delta_1)$ and $d_2 = d_2(\delta_2)$ be the lengths of the randomness strings for the decoding of GGHKR and $\mathcal{C}_{\mathsf{c}}$, so $d_1 = O(\log(k/\delta))$ and $d_2 = O(\log(N/\delta)) = O(\log(k/\delta)))$. Let $L = \widetilde{O}(\log(1/\delta)/\varepsilon^2)$ be as in Lemma 5.22. Given a randomness string $y = (y_1, y_2) \in \{0,1\}^{d_1+d_2}$, and $j \in [L]$, we generate a decoding circuit $C_{y,j}$ that gets input $i \in [k]$ and oracle access to $w \in \{0,1\}^n$, as follows.

- Use y_1 to generate the local decoder $C^1_{y_1}$ for GGHKR, of size s_1, making at most Q_1 non-adaptive queries. Use y_2 to generate the decoder $C^2_{y_2,j}$ for $\mathcal{C}_{\mathsf{c}}$, of size s_2, making at most Q_2 queries.
- Whenever $C^1_{y_1}$ wishes to query some index $z \in [k']$, we run the approximate local list decoder of $\mathcal{C}_{\mathsf{c}}$, namely $C^2_{y_2,j}(z)$, having query access to w.

The fact that each $C_{y,j}$ is a TC^0 circuit generated in space $O(\log(k/\varepsilon\delta))$, given (y,j), is immediate. Specifically, the size of each $C_{y,j}$ is bounded by

$$O(s_1 + Q_1 \cdot s_2) = \mathrm{polylog}(k/\delta) \cdot \frac{k^{c\gamma}}{\varepsilon^c}.$$

The total number of queries made is $Q_1 \cdot Q_2 = \mathrm{polylog}(k/\delta) \cdot \mathrm{poly}(1/\varepsilon)$.

For correctness, assume that $x \in \{0,1\}^k$ is such that w agrees with $\mathcal{C}(x)$ in at least $1/2+\varepsilon$ fraction of coordinates, and assume that y is good for both x and $\mathsf{GGHKR}(x)$, in the sense of Lemmas 5.4 and 5.22. We are guaranteed that for some $j^\star \in [L]$, $C^2_{y_2,j^\star}$ decodes correctly at least $1-\tau$ fraction of the symbols in $\mathsf{GGHKR}(x)$. Thus, when the local unique decoder is given the word $(C^2_{y_2,j^\star}(1),\ldots,...,C^2_{y_2,j^\star}(k'))$ as its noisy codeword, it essentially queries a word with $1-\tau$ agreement with $\mathsf{GGHKR}(x)$ and so $C^1_{y_1}$ correctly decodes every bit in x. The error probability over the y-s, due to non-adaptivity, follows from a simple union bound. We thereby established the desired properties of $\mathcal{C}$, which are summarized in Theorem 5.1.

For $\mathcal{C}_{\mathsf{LE}}$, we apply the same reasoning and combine the code $\mathsf{GGHKR}_{\mathsf{LE}}$ from Lemma 5.5 with $\mathcal{C}_{\mathsf{c}}$ from Lemma 5.22. Again, we note that this time we let the $\mathbf{TC}^0$ circuits make the queries to the message, rather then let the decoding algorithm that prints them do that. Moreover, *locality* is preserved.

Claim 5.23 (circuit complexity of $\mathcal{C}_{\mathsf{LE}}$ – I)**.** *There exists an algorithm that outputs a $\mathbf{TC}^0$ circuit of size $s = \mathrm{poly}(d, \log k, 1/\varepsilon, \log(1/\delta))$ that runs in space $O(\log s)$, and on input $i \in [n]$, returns the d coordinates $q_1,\ldots,q_D$ to be queried, where $D = d \cdot \mathrm{poly}(1/\varepsilon, \log(1/\delta))$. (That is, for all x, $\mathcal{C}_{\mathsf{LE}}(x)_i$ only depends on $x_{q_1},\ldots,x_{q_D}$.)*

Proof. The index $i \in [n]$ induces an index i' such that $\mathcal{C}_{\mathsf{LE}}(x)_i$ only depends on $(\mathsf{IW} \circ \mathsf{GGHKR}_{\mathsf{LE}})(x)_{i'}$, and the transformation $i \mapsto i'$ can be done by standard arithmetic of integers with $O(\log k)$ bits, and in particular using logspace-uniform $\mathbf{TC}^0$ circuits of size $\mathrm{polylog}(k)$. Now, each coordinate of IW depends on $Q = \mathrm{poly}(\log(1/\delta), 1/\varepsilon)$ locations in the codeword $\mathsf{GGHKR}_{\mathsf{LE}}(x)$ that are specified by $\overline{\mathsf{Loc}}(i',\cdot)$. By Claim 5.20, in space $O(\log(k/\varepsilon\delta) + \mathrm{loglog}(1/\delta))$ we can output a $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(k, 1/\varepsilon, \log(1/\delta))$ that computes those locations.

Now, each one of those locations gives rise to d locations to be queries from x, as implied by Claim 5.8. Composition of space-bounded algorithms gives us the desired algorithm that produces the $\mathbf{TC}^0$ circuit outputting the locations to be read, at the allotted size. ∎

Next, we need to establish the encoding step itself.

Claim 5.24 (circuit complexity of $\mathcal{C}_{\mathsf{LE}}$ – II)**.** *There exists a logspace-uniform $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(d, \log k, \log|\mathbb{F}|, \log(1/\delta), 1/\varepsilon)$ that given the above $x_{q_1},\ldots,x_{q_D} \in \{0,1\}$, outputs $\mathcal{C}_{\mathsf{LE}}(x)_i$.*

Proof. Note that the encoding of $\mathcal{C}_{\mathsf{c}}$ can be done by circuits of size $s_2 = \mathrm{poly}(\log k, 1/\varepsilon)$ generated in $O(\log(k/\varepsilon))$ space, and recall that given the corresponding coordinates of x, each coordinate of $\mathsf{GGHKR}_{\mathsf{LE}}(x)$ can be computed by circuits of size $s_1 = \mathrm{poly}(d, \log|\mathbb{F}|)$ generated in space $O(\log d + \mathrm{loglog}|\mathbb{F}|)$. Altogether, the size of a $\mathbf{TC}^0$ circuit that computes each coordinate of $\mathcal{C}_{\mathsf{LE}}(x)$ (the d coordinates of $\mathsf{GGHKR}_{\mathsf{LE}}$ can be computed in parallel) is thus

$$O(s_2 + Q \cdot s_1) = \mathrm{poly}(d, \log k, \log|\mathbb{F}|, \log(1/\delta), 1/\varepsilon)$$

and can be generated in space $O(\log d + \mathrm{loglog}|\mathbb{F}| + \log(k/\varepsilon)) = O(\log(k/\varepsilon))$. ∎

Finally, we claim that $\mathcal{C}_{\mathsf{LE}}$ is systematic. According to the standard definition, a code $\mathcal{C}$ is systematic if $\mathcal{C}(x)$ contains x itself as its prefix. Here, we obtain a somewhat weaker property that we describe next.

Claim 5.25 ($\mathcal{C}_{\mathsf{LE}}$ is weakly-systematic)**.** *There exists a logspace-uniform* $\mathbf{AC}^0$ *circuit of size* $\mathrm{poly}(d, \log k, \log|\mathbb{F}|)$ *such that given* $i \in [k]$ *and* $j \in [\log|\mathbb{F}|]$*, it outputs* $i' \in [n]$ *such that for all* $x \in \mathbb{F}^k$*,* $\mathcal{C}_{\mathsf{LE}}(x)_{i'}$ *is equal to the* j^{th} *bit in the encoding of* x_i*.*

Proof. First, note that the low-degree extension encoding, and the Hadamard encoding, are systematic in the standard sense. Also,

- The IW code maps f to $\bar{f}$ such that the i-coordinate of the symbol $\bar{f}_{\bar{z}}$ is given by $\mathsf{Samp}(z_1, i) \oplus \mathsf{Des}(z_2, i)$, where $\bar{z} = (z_1, z_2)$ and suitably instantiated sampler and design generator. We can assume, without any substantial change in parameters, that each $\mathsf{Samp}(z_1, \cdot)$ has an additional edge (say labeled by 1), mapping z_1 to its prefix of the appropriate length. Then, $f_{i'}$ can be found in $\bar{f}$ in the first coordinate of the symbol indexed by $(i' \circ \bar{0}, \bar{0})$. Clearly, this trivial mapping can be done in $\mathbf{AC}^0$.

- The ABNNR code maps $x \in \mathbb{F}$ to $\bar{x} \in (\mathbb{F})^d$ by aggregating symbols according to a balanced bipartite expander whose neighbor function, and its inverse, are computable by logspace-uniform $\mathbf{AC}^0$ circuits. Specifically, given an index i of x (or an index to a specific bit in the field element representation), we can compute $\Gamma(i, 1)$ and determine the location of x_i in the symbol $\bar{x}_{\Gamma(i,1)}$ by logspace-uniform $\mathbf{AC}^0$ circuits.

Our code $\mathcal{C}_{\mathsf{LE}}$ is constructed via (constantly many) compositions and concatenations of the above codes, instantiated with varying code lengths. Next, we observe that the weakly-systematic property is preserved under those operations. Indeed, if $\mathcal{C}_1$ and $\mathcal{C}_2$ are weakly systematic, then the composition $\mathcal{C}_2 \circ \mathcal{C}_1$ is also weakly systematic simply by applying the mapping circuit for $\mathcal{C}_2$ and then the mapping circuit for $\mathcal{C}_1$. In a very similar manner, the property is preserved under concatenating an outer code $\mathcal{C}_1$ with an inner code $\mathcal{C}_2$, and this concludes our claim. ■

The properties of $\mathcal{C}_{\mathsf{LE}}$ are summarized in Theorem 5.2 above.

6 Deducing BPL $=$ L from Uniform Hardness Assumptions

We begin by setting up notation. Fix a family $\{C_n\}$ of threshold circuits of size $T = T(n)$ and depth $d = d(n)$. Since we will be concerned with constant depth $d = O(1)$, we assume for simplicity that each of the d layers has exactly T threshold gates. For any $n \in \mathbb{N}$, $i \in [d]$, and $j \in [T] \times [T]$, let $g_{i,j}$ be the j^{th} gate in the i^{th} layer of C_n, and denote

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

where $w_{i,j,k} \in \{-T, ..., T\} \subseteq \mathbb{Z}$ and $\theta_{i,j} \in \{-T^2, ..., T^2\}$. (The notations $g_{i,j}, w_{i,j,k}, \theta_{i,j}$ do not explicitly refer to the circuit family $\{C_n\}$ or to the input length n, but these will be clear from context.) Indeed, we assume that all weights are integers with absolute value at most T, and we bound the threshold values accordingly.

Our hardness-vs.-randomness tradeoff will use hard functions computable in uniform TC^0, where the precise uniformity condition is as follows.

Definition 6.1 (logspace-uniform threshold circuits)**.** *We say that a family of threshold circuits of size T and depth d is* logspace-uniform *if:*

1. *There is a machine* Weight *that gets input $(1^n, i, j, k)$ where $i \in [d]$ and $(j,k) \in [T]^2$, runs in space $O(\log T)$, and prints $w_{i,j,k}$.*
2. *There is a machine* Thr *that gets input $(1^n, i, j)$ where $i \in [d]$ and $j \in [T]$, runs in space $O(\log T)$, and prints $\theta_{i,j}$.*

Lemma 6.2 (canonical form for logspace-uniform threshold circuits)**.** *There are two universal constants $c, c' > 1$ such that the following holds for any space-computable $\delta = \delta(n) \in (0,1)$. Let $\{C_n\}$ be a logspace-uniform TC^0 circuit family of size $T = T(n)$ and depth $d = d(n)$. Then, there exists a logspace-uniform TC^0 circuit family $\{C'_n\}$ of size $T' = T^c$ and depth $d' = c \cdot (d/\delta)$ that computes the same function as $\{C_n\}$, and that satisfies the following:*

1. *The bottom layer of C'_n has $n + B_g$ gates, where $B_g = \widetilde{O}(T^2)$. The first n gates are input gates $x_1, ..., x_n$, and the last B_g gates are constant gates (i.e., with fan-in zero). There is a machine running in space $O(\log T)$ that gets as input $i \in [B_g]$, and prints the type of the $(i+n)^{th}$ gate at the bottom layer (i.e., it prints either the constant zero or the constant one).*
2. *The $d' - 1$ layers above the bottom layer have unweighted majority gates of fan-in $T^{c \cdot \delta}$. There is a machine that gets input 1^n, runs in time $\mathrm{polylog}(T)$ and space $O(\log T)$, and prints a formula that decides the following problem: Given input $(i,j,k) \in [d'-1] \times [T] \times [T]$, output one if gate k in layer $i-1$ feeds into gate j in layer i, and zero otherwise.*

Proof. We transform C_n into C'_n in two steps. In the first step, we consider D_n that has $n + B_g$ gates at the bottom layer: The first n are input gates, and the last $B_g = \widetilde{O}(T^2)$ represent the description of C_n. Now, let $U_{n,d}$ be a universal TC^0 circuit of depth $O(d)$ and size $\mathrm{poly}(T)$ that simulates a TC^0 circuits of depth d and size T, on inputs of length n. In the $O(d)$ layers above the bottom layer, the circuit D_n simulates $U_{n,d}$ on input (C_n, x).

Note that there is a space-$O(\log T)$ machine printing the type of the non-input gates at the bottom layer. Also, since $U_{n,d}$ has a very simple structure, there is a formula that can be printed in time $\mathrm{polylog}(T)$ and space $O(\log T)$ (in particular, the formula is of size at most $\mathrm{polylog}(T)$) that decides the connectivity between gates in $U_{n,d}$.[26]

[26]To see this, recall that $U_{n,d}(C_n, x)$ works in d stages, where each stage i' computes the gate values of $C_n(x)$ at layer i'. Denoting the gate values of layer $i'-1$ by $h_1, ..., h_T$, the functionality of $U_{n,d}$ is as follows: For each $j' \in [T]$, multiply each $h_{k'}$ by $w_{i',j',k'}$, where $w_{i',j',k'}$ appears in the description of C_n; compute the

Next, we transform D_n into a circuit with unweighted majority gates of fan-in $T^{O(\delta)}$. To do this, we simulate each gate g (in the layers above the bottom layer) by a sub-circuit of depth $O(1/\delta)$. Assume that $g(h_1, ..., h_T) = \mathbf{1}\left[\sum_{i\in[T]} w_i \cdot h_i \geq \theta\right]$. The sub-circuit first computes, for each $i \in [T]$, the mapping $h_i \mapsto w_i \cdot h_i$. Then it computes the summation $\sum w_i \cdot h_i$ in $O(1/\delta)$ stages, at each stage adding T^δ integers. And finally, it compares the computed sum $\sum w_i \cdot h_i$ to the threshold θ.

Recall that multiplication, iterated addition, and comparison, can all be performed by $\mathbf{TC}^0$ circuits with very simple structure. Thus, there is a uniform formula that can be printed in time $\mathrm{polylog}(T)$ and space $O(\log T)$ describing the connectivity in the layers above the bottom one. ■

Organization. In Section 6.1 we construct a (reconstructive) *targeted somewhere-PRG*, which will be the main technical component in the proof of Theorem 1. In Section 6.2 we prove Theorem 1 using this targeted somewhere-PRG. In Section 6.3, we prove Proposition 1.2, which asserts unconditional lower bounds in $\mathbf{L}$ for log-spaceadvice-uniform $\mathbf{TC}^0$ circuits with oracle access to bounded-space machines (recall that log-spaceadvice-uniform $\mathbf{TC}^0$ circuits were defined in Definition 1.1). In Section 6.4 we prove Theorem 2, which asserts a hardness-vs.-randomness tradeoff for linear space (from worst-case and fully uniform hardness assumptions). Lastly, in Section 6.5, we show that average-case derandomization of $\mathbf{BPL}$ with zero error reduces to standard average-case derandomization of $\mathbf{BPL}$.

---

sum $\sum_{k'} w_{i',j',k'} h_{k'}$; then compare the sum to $\theta_{i',j'}$, which appears in the description of C_n.

Let us sketch the proof of how connectivity in $U_{n,d}$ can be decided by a formula that can be generated in space $O(\log T)$ and time $\mathrm{polylog}(T)$. The formula is given (i, j, k), where j is the index of a gate g in layer i of $U_{n,d}$, and k is the index of gate h in layer $i-1$ of $U_{n,d}$. The formula parses $(i, j, k) = ((i', j', b, k'), (j'_0, k'_0))$.

The indices (i', j') indicate that gate g is part of the simulation of gate j' in layer i' of the input circuit to $U_{n,d}$. The index $b \in [3]$ indicates which of the three parts of siluating j' g is part of; that is, whether g is in a sub-circuit computing multiplication (i.e., $w_{i',j',k'} \cdot h_{k'}$ for some $k' \in [T]$), the sub-circuit computing iterated addition, or the sub-circuit computing comparison to $\theta_{i',j'}$. The index k' is used only when b is computing multiplication, in which case it indicates that g is part of the sub-circuit computing $w_{i',j',k'} \cdot h_{k'}$. The indices (j'_0, k'_0) indicate the locations of g and of h within this sub-circuit.

The parsing above reduces the problem of deciding connectivity in $U_{n,d}$ to the problem of deciding connectivity of (j'_0, k'_0) in a circuit that implements multiplication, iterated addition, or comparison to a fixed value. The only additional cost in the reduction is computing the location of $w_{i',j',k'}$ or of $\theta_{i',j'}$ on the input tape to $U_{n,d}$. Thus, perfoming the reduction only requires computing simple arithmetic operations on (i, j, k), which can be done by a formula with the claimed complexity.

The claim follows by combining this reduction with the fact that connectivity in each of the sub-circuits (i.e., for multiplication, iterated addition, or comparison) can be decided by a formula that can be printed in space $O(\log T)$ and time $\mathrm{polylog}(T)$. This is because the standard constructions of $\mathbf{TC}^0$ circuits for all of these operations are very simple, and the connectivity in the $\mathbf{TC}^0$ circuits can be decided by simple arithmetic operations on (j'_0, k'_0).

6.1 A Reconstructive Targeted Somewhere-PRG

Our goal in this section is to construct a reconstructive targeted somewhere-PRG that is based on a function in logspace-uniform TC^0 and whose reconstruction is in *deterministic* logspace-uniform TC^0.

To do so, in Section 6.1.1 we show that any function in logspace-uniform TC^0 admits a very efficient *bootstrapping system*, which is a notion we will define in that section. Then, in Section 6.1.2 we construct the targeted somewhere-PRG, which can be based on any function with that very efficient bootstrapping system.

6.1.1 Bootstrapping systems for logspace-uniform threshold circuits

We first define a *polynomial decomposition* of a threshold circuit. The definition follows ideas from [CTW23], which in turn is based on [GKR15; CT21b].

At a high level, a polynomial decomposition of a circuit $C_n(x)$ (i.e., for a fixed input x) is a sequence of polynomials that represent the values of the gates in $C_n(x)$ and that is "downward self-reducible" (i.e., computing a polynomial in the sequence at any point reduces to computing the preceding polynomial in the sequence at "a few" points). In more detail, for each layer i, we introduce a polynomial $\hat{\alpha}_i$ that is an arithmetization of the sequence of gate-values of the i^{th} layer of $C_n(x)$. We then introduce $2m = O(1)$ intermediary polynomials between each pair $\hat{\alpha}_i$ and $\hat{\alpha}_{i+1}$ (for a carefully chosen constant m), denoted $\hat{\alpha}_{i+1,0}, ..., \hat{\alpha}_{i+1,2m}$, such that computing $\hat{\alpha}_{i+1,0}$ efficiently reduces to computing $\hat{\alpha}_i$, and computing $\hat{\alpha}_{i+1,j+1}$ efficiently reduces to computing $\hat{\alpha}_{i+1,j}$, and $\hat{\alpha}_{i+1,2m} = \hat{\alpha}_{i+1}$.

To arithmetize TC^0 in this manner, we will actually define each $\hat{\alpha}_i$ to be not the encoding of the gate-values in the i^{th} layer of $C_n(x)$, but the encoding of the sequence $\{\sigma_g(x)\}_{g\text{-s in the } i^{\text{th}} \text{ layer}}$ where $\sigma_g(x)$ is the sum that underlies the threshold gate g (i.e., if $g(x) = \mathbf{1}[\sum_j w_{g,j} \cdot h_j(x) > \theta_g]$, then $\sigma_g = \sum_j w_{g,j} \cdot h_j(x)$).

We will later on argue that every logspace-uniform family of TC^0 circuits admits a polynomial decomposition that is *efficient*: The polynomials have low-degree, and all reductions are indeed efficiently computable.

Definition 6.3 (polynomial decomposition of a threshold circuit)**.** *Let C be a circuit that has n input bits, size T, depth d, and unweighted majority gates of fan-in φ. For every $x \in \{0,1\}^n$, we call a collection of polynomials a* polynomial decomposition of $C(x)$ *if it meets the following specifications.*

1. ***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 $H = [h] \subseteq \mathbb{F}$, and 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.*[27]

[27]If $\vec{u}$ is not in the range of ξ then $\xi^{-1}(\vec{u}) = \perp$. We always use ξ to encode an index i as an element from H^m. We will pick a 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 TC^0

2. ***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* $w_{i,j,k}$*.*[28] *The polynomial* $\hat{\Phi}_i \colon \mathbb{F}^{2m} \to \mathbb{F}$ *can be any extension of* Φ_i*.*

3. ***Input polynomial.*** *Let* $\alpha_0 \colon H^m \to \{0, 1\}$ *represent the bottom layer of* $C_n(x)$ *(i.e., with* x *placed at the values of input gates of* C_n*), padded with* 0*-s to be of length* h^m*.*[29] *Let* $\hat{\alpha}_0 \colon \mathbb{F}^m \to \mathbb{F}$ *be the standard Lagrange interpolation of* α_0*, defined by*

$$\hat{\alpha}_0(\vec{u}) = \sum_{\vec{z} \in H^m} \delta_{\vec{z}}(\vec{u}) \cdot \alpha_0(\vec{z}) \ ,$$

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

4. ***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).*[30] *We define polynomials* $\hat{\alpha}_i \colon \mathbb{F}^m \to \mathbb{F}$ *as follows:*

$$\hat{\alpha}_1(\vec{u}) = \sum_{\vec{v} \in H^m} \hat{\Phi}_1(\vec{u}, \vec{v}) \cdot \hat{\alpha}_0(\vec{v})$$

$$\hat{\alpha}_i(\vec{u}) = \sum_{\vec{v} \in H^m} \hat{\Phi}_i(\vec{u}, \vec{v}) \cdot \delta_{>\theta}(\hat{\alpha}_{i-1}(\vec{v})) \ , \quad i \in \{2, ..., d\} \ .$$

Above, $\theta = \lfloor \varphi/2 \rfloor$ *and* $\delta_{>\theta}$ *is a polynomial of degree* $\varphi - 1$ *that maps every* $a \in [\varphi]$ *to* $\delta_{>\theta}(a) = \begin{cases} 1 & a > \theta \\ 0 & o.w. \end{cases}$*.*[31]

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

$$\hat{\alpha}_{1,0}(\vec{u}, \sigma_1, ..., \sigma_m) = \hat{\Phi}_1(\vec{u}, \sigma_{1,...,m}) \cdot \hat{\alpha}_0(\sigma_{1,...,m})$$

$$\hat{\alpha}_{i,0}(\vec{u}, \sigma_1, ..., \sigma_m) = \hat{\Phi}_i(\vec{u}, \sigma_{1,...,m}) \cdot \delta_{>\theta}(\hat{\alpha}_{i-1}(\sigma_{1,...,m})) \ , \ \ i \in \{2, ..., d\}$$

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

$$\hat{\alpha}_{1,j}(\vec{u}, \sigma_1, ..., \sigma_{m-j}) = \sum_{\sigma_{m-j+1}, ..., \sigma_m \in H} \hat{\Phi}_1(\vec{u}, \sigma_{1,...,m}) \cdot \hat{\alpha}_0(\sigma_{1,...,m})$$

$$\hat{\alpha}_{i,j}(\vec{u}, \sigma_1, ..., \sigma_{m-j}) = \sum_{\sigma_{m-j+1}, ..., \sigma_m \in H} \hat{\Phi}_i(\vec{u}, \sigma_{1,...,m}) \cdot \delta_{>\theta}(\hat{\alpha}_{i-1}(\sigma_{1,...,m})) \ , \ \ i \in \{2, ..., d\}$$

where $\sigma_{k,...,k+r} = \sigma_k, \sigma_{k+1}, ..., \sigma_{k+r}$*. Observe that* $\hat{\alpha}_{i,m} \equiv \hat{\alpha}_i$*.*

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

[29] Recall that, as in Lemma 6.2, there may be gates at the bottom layer of C_n that are not input gates, so the padding of zeroes only appears after those bits.

[30] Formally, for every $\vec{u} \in H^m$ we have $\alpha_i(\vec{u}) = \begin{cases} g_{i,\xi^{-1}(\vec{u})} & \xi^{-1}(\vec{u}) \neq \perp \\ 0 & \text{o.w.} \end{cases}$.

[31] That is, $\delta_{>\theta}(a) = \sum_{\sigma \in [\varphi]} \prod_{\sigma' \in [\varphi] \setminus \{\sigma\}} \frac{a - \sigma'}{\sigma - \sigma'} \cdot \mathbf{1}[\sigma > \theta]$.

We now argue that logspace-uniform threshold circuits (from Definition 6.1) have very efficient polynomial decompositions. To be exact, we transform any such circuit family into a family with unweighted majority gates of bounded fan-in (using Lemma 6.2) and argue that the latter family has a suitable polynomial decomposition. In the result statement below, we use the same notation and definition as in Definition 6.3 (in particular, the same definitions of the $\hat{\alpha}_i$-s and $\hat{\alpha}_{i,j}$-s).

Proposition 6.4 (efficient polynomial decompositions of logspace-uniform threshold circuits)**.** *There exists a universal constant $c \in \mathbb{N}$ such that the following holds. Let $\{C_n\}$ be a logspace-uniform family of circuits of size $T = T(n)$ and depth $d = d(n)$, and let $\delta \in (0,1)$ be a constant. Then, there is a logspace-uniform family of circuits $\{C'_n\}$ of size $T' = T^c$ and depth $d' = c \cdot (d/\delta)$ computing the same function as $\{C_n\}$, such that for every $x \in \{0,1\}^n$, there exists a polynomial decomposition of $C'_n(x)$ satisfying:*

1. ***Arithmetic setting.*** *The polynomials are defined over $\mathbb{F} = \mathbb{F}_p$, where p is the smallest prime in the interval $[5 \cdot (T')^2 + 1,\ 10 \cdot (T')^2]$. Let $H = [h] \subseteq \mathbb{F}$, where h is the smallest power of two of magnitude at least $(T')^{\delta/3}$, and let m be the minimal integer such that $h^m \geq 2T'$.*

2. ***Faithful representation.*** *For every $i \in [d']$ and $\vec{u} \in H^m$ representing a gate in the i^{th} layer of C'_n, the value of the gate in $C'_n(x)$ is 1 if and only if $\hat{\alpha}_i(\vec{u}) \geq \theta_{i,\vec{u}}$.*[32]

3. ***Low degree.*** *All polynomials in the polynomial decomposition except for $\hat{\alpha}_0$ have total degree at most $T^{c \cdot \delta}$.*

4. ***Base case.*** *There is a machine B that gets input 1^n and $i \in [B_g]$ where $B_g = \widetilde{O}(T^2)$, runs in space $O(\log T)$, and outputs an element in $\mathbb{F}$ so that the following holds. There is a logspace-uniform TC^0 circuit of size $(n \cdot h)^c$ that get input $\vec{v}$, and non-adaptive oracle access to the fixed input x and to B, and outputs $\hat{\alpha}_0(\vec{v})$.*

   (Recall that the decomposition is defined with respect to any fixed input x. We stress that the circuit for $\hat{\alpha}_0$ has oracle access to this x, but the machine B does not. Indeed, the behavior of B does not depend on x, but only on the family $C = \{C_n\}$.)

5. ***Downward self-reducibility.*** *There is a machine S that gets as input $2m$ elements of $\mathbb{F}$, and an advice φ of length $\mathrm{polylog}(T)$, runs in space $c \cdot \delta \cdot \log T$, and outputs an element of $\mathbb{F}$. There are two logspace-uniform non-adaptive oracle TC^0 circuits of size h^c 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 $\hat{\alpha}_{i-1}$ and to S, output $\hat{\alpha}_{i,0}(\vec{u}, \sigma_1, ..., \sigma_m)$.[33]

   (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 $\hat{\alpha}_{i,j-1}$ and to S, output $\hat{\alpha}_{i,j}(\vec{u}, \sigma_1, ..., \sigma_{m-j})$.

[32] The notation $\theta_{i,\vec{u}}$ refers to the threshold value of gate $\vec{u}$ in the i^{th} layer of C'_n. To avoid confusion, we note in advance that C'_n will only have unweighted majority gates of fixed fan-in φ, and thus $\theta_{i,\vec{u}} = \lfloor \varphi/2 \rfloor$ regardless of $(i, \vec{u})$.

[33] Having oracle access to a polynomial $\hat{\alpha}_i$ means being able to send a query $\vec{v}$ and receive answer $\hat{\alpha}_i(\vec{v})$.

Moreover, the advice $\varphi = \varphi(n)$ can computed in space $O(\log T)$.

Proof. The circuit family $\{C'_n\}$ is obtained from Lemma 6.2. Denote its size by T' and its depth by d'. To define the polynomial decomposition we need to specify the extensions of the circuit-structure functions Φ_i. We will do so relying on the following claim.

Claim 6.4.1. *For every $i \in [d']$ there exists $\hat{\Phi}_i \colon \mathbb{F}^{2m} \to \mathbb{F}$ that satisfies the following:*

1. *For every $(\vec{u}, \vec{v}) \in H^{2m}$ it holds that $\hat{\Phi}_i(\vec{u}, \vec{v}) = 1$ if gate $\vec{u}$ in the i^{th} layer is fed by gate $\vec{v}$ in the $(i-1)^{th}$ layer, and $\hat{\Phi}_i(\vec{u}, \vec{v}) = 0$ otherwise.*

2. *The degree of $\hat{\Phi}_i$ is at most $h \cdot \mathrm{polylog}(T)$.*

3. *There is a machine that gets input $(i, \vec{u}, \vec{v})$ and an advice $\varphi \in \{0,1\}^{\mathrm{polylog}(T)}$, runs in space $c_1 \cdot \log(h)$ for a universal constant $c_1 > 1$, and outputs $\hat{\Phi}_i(\vec{u}, \vec{v})$. Furthermore, the advice $\varphi = \varphi(n)$ can be computed from input 1^n in space $O(\log T)$.*

Proof. Recall that $(i, j, k) \mapsto \Phi_i(j, k)$ is computable by a formula that can be printed in time $\mathrm{polylog}(T)$ and space $O(\log T)$. We let φ be the description of that formula. Consider Φ_i as a function $\mathbb{F}_2^{2\log(T')} \to \mathbb{F}_2$, and observe that it is computable by an arithmetic formula of degree $\mathrm{polylog}(T)$ whose structure mimics φ. For each i, this yields an arithmetic formula computing a polynomial $\Phi'_i \colon \mathbb{F}^{2\log(T')} \to \mathbb{F}$ of degree $\mathrm{polylog}(T)$ that agrees with Φ_i on $\mathbb{F}_2^{2\log(T')}$.[34]

Now we want to construct a polynomial that gets inputs in $(\vec{u}, \vec{v}) \in \mathbb{F}^{2m}$, "projects" each element in $\vec{u}$ and in $\vec{v}$ to its binary representation (i.e., over $\mathbb{F}_2$), and computes Φ'_i on the resulting sequence of $\mathbb{F}_2$-elements. Since we only care about the behavior of this polynomial on inputs $(\vec{u}, \vec{v}) \in H^{2m}$, it suffices to consider a binary representation of length $\ell = \log(h)$, in which case the complexity of this operation is low enough. In more detail, for every $j \in [\ell]$ consider $\pi_j \colon H \to \{0,1\}$ such that $\pi_j(a)$ is the j^{th} bit in the binary representation of a. Note that there is a polynomial $\hat{\pi}_j \colon \mathbb{F} \to \mathbb{F}$ of degree at most h that agrees with π_j on H. Finally, let $\hat{\Phi}_i \colon \mathbb{F}^{2m} \to \mathbb{F}$ such that

$$\hat{\Phi}_i(z_1, ..., z_{2m}) = \Phi'_i(\hat{\pi}_1(z_1), ..., \hat{\pi}_\ell(z_1), ..., \hat{\pi}_1(z_{2m}), ..., \hat{\pi}_\ell(z_{2m})) \ .$$

Note that $\hat{\Phi}_i$ is of degree $h \cdot \mathrm{polylog}(T)$ and that it agrees with Φ_i on $H^m \times H^m$. We show that $\hat{\Phi}_i$ is computable by a machine M that runs in space $(c_1 \cdot \log(h))$ and gets $\mathrm{polylog}(T)$ bits of advice, where the advice can be computed from input 1^n in space $O(\log T)$.

The advice φ is simply the description of Φ'; as argued above, it is of length $\mathrm{polylog}(T)$ and can be generated in space $O(\log T)$. The machine M gets input $(\vec{u}, \vec{v})$ and runs the

[34] Specifically, let F be the Boolean formula computing $(i, j, k) \mapsto \Phi_i(j, k)$. We simulate F by an arithmetic formula F' that replaces every Boolean gate $g(h_1, h_2)$ in F by a constant-sized arithmetic gadget computing the same function over $\mathbb{F}_2$ (e.g., AND is computed by a multiplication gate, and OR is computed by $1+$AND). The formula F' has essentially the same complexity as F, up to constant factors. Hard-wiring i, we get a polynomial $\Phi'_i \colon \mathbb{F}^{2m} \to \mathbb{F}$, and the total degree of this polynomial is at most the size of the formula computing it (recall that a formula of size s computes a polynomial of total degree at most s).

DFS-style simulation of Φ', computing it as an arithmetic formula over $\mathbb{F}$ (for a careful implementation of the DFS-style simulation on formulas of depth that can be super-logarithmic in their size, see e.g., [CDS+23, Lemma 6.13]). Whenever Φ' accesses one of its inputs $\hat{\pi}_i(z_j)$, the machine M computes $\hat{\pi}_i$ at z_j via Lagrange interpolation (i.e., $\hat{\pi}_i(u) = \sum_{a \in H} \pi_j(a) \cdot \prod_{a' \in H \setminus \{a\}} \frac{u-a'}{a-a'}$). The DFS-style simulation has $O(\operatorname{loglog} T)$ levels (the size of the formula for Φ_i is $\operatorname{polylog}(T)$, and thus its depth is $O(\operatorname{loglog} T)$), and in each path there one input $\hat{\pi}_i(z_j)$ that is read. The space complexity of M is dominated by computing the $\hat{\pi}_i$-s, and thus M runs in overall space at most $O(\log h)$.

□

The extensions $\hat{\Phi}_i \colon \mathbb{F}^{2m} \to \mathbb{F}$ are the ones given by Claim 6.4.1, and they suffice to fully determine the polynomial decomposition, i.e., the $\hat{\alpha}_i$-s and the $\hat{\alpha}_{i,j}$-s. We now verify that the decomposition has the required properties.

The *faithful representation* property follows from the fact that Φ'_i agrees with Φ_i on H^{2m}, and by the definitions of the $\hat{\alpha}_i$-s. (To see this, argue by induction that for each layer $i = 1, \ldots, d'$, the following holds: For each $\vec{u} \in \mathbb{F}^m$ we have that $\hat{\alpha}_i(\vec{u})$ is the sum underlying gate $\vec{u}$.[35] For $i = 1$ this holds by the definition of $\hat{\alpha}_1$, and for $i > 1$ we use the induction hypothesis and the definition of $\delta_{>\theta}$. In both the base case and the inductive step, we used the fact that Φ'_i agrees with Φ_i on H^{2m}.) The *degree bound* property follows from the degree of the $\hat{\Phi}_i$-s and by the fact that $\delta_{>\theta}$ is of degree $T^{O(\delta)}$ (this is because the fan-ins in C'_n are at most $\varphi = T^{O(\delta)}$).

For the *base case* property, let B_0 be the machine printing the types of the last $B_g = \widetilde{O}(T^2)$ gates at the bottom layer, given to us in Lemma 6.2. Recall that B_0 runs in space $O(\log T)$ and does not depend on the input x. Now, recall that

$$\hat{\alpha}(\vec{u}) = \sum_{\vec{z} \in H^m} \delta_{\vec{z}}(\vec{u}) \cdot \alpha_0(\vec{z}) \ . \tag{6.1}$$

We partition H^m into three sets: A set X representing the n input gates, a set Z representing the additional B_g nontrivial gates, and an additional set $H^m \setminus (X \cup Z)$. Recall that for every $\vec{z} \in H^m \setminus (X \cup Z)$ we have that $\alpha_0(\vec{z}) = 0$. Thus, the sum in Equation (6.1) can be presented as

$$\hat{\alpha}(\vec{u}) = \sum_{\vec{z} \in X} \delta_{\vec{z}}(\vec{u}) \cdot x_{\xi^{-1}(\vec{z})} + \sum_{\vec{z} \in Z} \delta_{\vec{z}}(\vec{u}) \cdot B_0(1^n, \xi^{-1}(\vec{z}) - n) \ .$$

Observe that the function $(1^n, \vec{u}) \mapsto \sum_{\vec{z} \in Z} \delta_{\vec{z}}(\vec{u}) \cdot B_0(1^n, \xi^{-1}(\vec{z}) - n)$ can be computed in space $O(\log T)$, and does not depend on the input x. We define B to be the machine computing this function. Hence, given $\vec{u}$, we can compute Equation (6.1) by a logspace-uniform TC^0 circuit of size $\operatorname{poly}(n, h)$ that makes non-adaptive oracle queries to x, and a single non-adaptive oracle query to B.

[35] That is, if the gate g in layer i represented by $\vec{u}$ computes the function $g(x) = \mathbf{1}[\sum_j w_{g,j} h_j(x) \geq \theta]$, then $\hat{\alpha}_i(\vec{u}) = \sum_j w_{g,j} h_j(x)$.

Lastly, for the *downward self-reducibility* property, the machine S will be the one computing $\hat{\Phi}_i$, from Claim 6.4.1. By the definition of $\hat{\alpha}_{i,0}$, computing it reduces to computing $\delta_{>\theta}$ and $\hat{\Phi}_i$; the former can be done in logspace-uniform TC^0 of size h^c, and the latter can be done in space $O(\log h) = O(c \cdot \delta \cdot \log T)$ given the advice φ. Similarly, computing $\hat{\alpha}_{i,j}$ reduces to summing h computations of a form similar to that of $\hat{\alpha}_{i,0}$ (i.e., each of the summand reduces to computing $\hat{\Phi}_i$ and $\delta_{>\theta}$), and thus can also be done with a complexity overhead multiplicative in $\mathrm{poly}(h)$, compared to $\hat{\alpha}_{i,0}$. ■

Finally, we show that any function with a very efficient *polynomial decompositions* also has a very efficient *bootstrapping system*. Loosely speaking, a bootstrapping system for a circuit computation $C_n(x)$ is a sequence of functions that encode the layers of $C_n(x)$ (or, more accurately, the sequence contains encodings of the layers of $C_n(x)$, among other functions), that are "downward self-reducible" (i.e., computing a function efficiently reduces to computing the preceding function), and such that any function in the sequence can be efficiently reconstructed, i.e. if we can compute it efficiently on $1/2 + o(1)$ of the inputs, then we can compute it efficiently on all inputs.

The bootstrapping system will be obtained by combining the polynomial decompositions with the locally encodable codes from Theorem 5.2.

Proposition 6.5 (bootstrapping systems for logspace-uniform threshold circuits)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $\{C_n\}$ be a logspace-uniform family of TC^0 circuits of size $T = T(n)$ and constant depth $d = d(n)$, and let $\eta, \delta \in (0,1)$ be constants. Then, there is a constant $\kappa > 1$ that only depends on δ such that for every $x \in \{0,1\}^n$ there exists a sequence of functions $w_x^{(1)}, ..., w_x^{(\bar{d})} \colon [T^\kappa] \to \{0,1\}$, where $\bar{d} = c \cdot (d/\delta^2)$, satisfying the following:*

1. ***Faithful representation.*** *There is a logspace-uniform oracle TC^0 circuit family $\{\mathtt{OUT}_n\}$ of size $T^{c \cdot \delta}$ such that, when $\mathtt{OUT}_n$ is given $j \in [T]$ and oracle access to $w_x^{(\bar{d})}$, it outputs the value of the j^{th} output gate of $C'_n(x)$ (or 0, if the output of $C'_n(x)$ is of length less than j).*

2. ***Base case.*** *There is a machine B that gets input 1^n and $i \in [B_g]$, runs in space $O(\log T)$, and outputs an element in $\mathbb{F}$ so that the following holds. There is a logspace-uniform TC^0 circuit family $\{\mathtt{BASE}_n\}$ of size $(n \cdot T^\delta)^c$, such that, when $\mathtt{BASE}_n$ is given $i \in [T^\kappa]$ and non-adaptive oracle access to $x \in \{0,1\}^n$ and to B, outputs $w_x^{(1)}(i)$.*

3. ***Downward self-reducibility.*** *There is a machine S that gets as input $O(1/\delta)$ elements in $\mathbb{F}$ and an advice φ of length $\mathrm{polylog}(T)$, runs in space $c \cdot (\delta \log T)$, and outputs an element in $\mathbb{F}$ so that the following holds. There is a logspace-uniform oracle TC^0 circuit family $\{\mathtt{DSR}_{n,i}\}_{n \in \mathbb{N}, i \in \{2,...,d'\}}$ of size $T^{c \cdot \delta}$, such that, when $\mathtt{DSR}_{n,i}$ is given $j \in [T^\kappa]$ and non-adaptive oracle access to $w_x^{(i-1)}$ and to S, outputs $w_x^{(i)}(j)$.*

   Moreover, the advice $\varphi = \varphi(n)$ can be computed from input 1^n in space $O(\log T)$.

4. ***Deterministic layer reconstruction.*** *There is an algorithm that gets input 1^n, a seed y of length $O(\log T)$, and an index $i \in \{2, ..., d'\}$; the algorithm runs in space $O(\log T)$, and prints an oracle $\mathbf{TC}^0$ circuit $\mathtt{REC}_{n,y,i}$ of size $T^{c\cdot\delta}$ that satisfies the following.*

   (a) The circuit first makes non-adaptive queries to $w_x^{(i)}$, as a preprocessing step (that does not depend on its input). After preprocessing, it gets input $j \in [T^\kappa]$ and oracle access to $\mathcal{O}$, and outputs a bit.

   (b) Let $\mathcal{O}\colon [T^\kappa] \to \{0,1\}$ be such that

   $$\Pr_{j\in[T^\kappa]}[\mathcal{O}(j) = w_x^{(i)}(j)] \geq 1/2 + T^{-\delta}.$$

   Then, with probability at least $1-\eta$ over y it holds that $\mathtt{REC}^{\mathcal{O}}_{n,y,i} \equiv w_x^{(i)}$ (i.e., $\mathtt{REC}^{\mathcal{O}}_{n,y,i}(j) = w_x^{(i)}(j)$ for all $j \in [T^\kappa]$).

   Lastly, there exists an algorithm that gets input (x, i, j), where $x \in \{0,1\}^n$ and $i \in [\bar{d}]$ and $j \in [T^\kappa]$, runs in space $O(\log T)$, and outputs $w_x^{(i)}(j)$.

Proof. Let $c'' > 1$ be the universal constant from Proposition 6.4. We apply Proposition 6.4 to $\{C_n\}$, to obtain a family $\{C'_n\}$ such that for every $x \in \{0,1\}^n$, $C'_n(x)$ has a polynomial decomposition with polynomials of degree $\Delta = T^{c''\cdot\delta}$. We reindex the polynomials into a sequence $\{P_i\}_{i\in[\bar{d}]}$, where $\bar{d} = d' \cdot (m+1) + 1$, using the following ordering:

$$\underbrace{\hat{\alpha}_0}_{P_1}, \underbrace{\hat{\alpha}_{1,0}}_{P_2}, ..., \hat{\alpha}_{1,m}, \hat{\alpha}_{2,0}, ..., \hat{\alpha}_{2,m}, \hat{\alpha}_{d',1}, ..., \underbrace{\hat{\alpha}_{d',m}}_{P_{\bar{d}}}.$$

Recall that $d' = O(d/\delta)$ and that $m = O(1/\delta)$, and thus $\bar{d} = O(d/\delta^2)$. For convenience, we add dummy variables to the polynomials, so that they all map $\mathbb{F}^{2m} \to \mathbb{F}$ (note that this does not affect any of the properties claimed in Proposition 6.4).

For $i \in \{2, ..., \bar{d}\}$, we identify P_i with the string $P_i \in \mathbb{F}^{p^{2m}}$ representing its evaluations on all inputs. We use the code Enc from Theorem 5.2 with parameters:

- $k = p^{2m}$,
- $d = (100\cdot\Delta)^{2/c'} = T^{O_{c'',c'}(\delta)}$, where $c' > 1$ is the universal constant from Theorem 5.2,
- $\delta = \eta/4$,
- $\varepsilon = T^{-\delta}$, and,
- $\gamma = \delta^2$.

Note that with these parameters, the output length of the code is $\mathrm{poly}_\gamma(p^{2m}/\varepsilon) = T^\kappa$, for a sufficiently large constant $\kappa > 1$. (Also note that the hypothesis in Theorem 5.2 that $|\mathbb{F}| = p$ is at most exponential in $k = p^{2m}$ is indeed satisfied.)

For $i \geq 2$, we define $w_x^{(i)} \colon [T^\kappa] \to \{0,1\}$ so that

$$w_x^{(i)}(j) = \mathsf{Enc}(P_i)_j \ .$$

For $i = 1$, we define $w_x^{(1)} \colon [T^\kappa] \to \{0,1\}$ that represents $\hat{\alpha}_0$ without the encoding Enc. Specifically, any input $j \in [T^\kappa]$ is parsed as (j_0, k, ℓ) where $j_0 \in [p^{2m}]$ represents a vector $\vec{u} \in \mathbb{F}^{2m}$, and $k \in [\lceil \log p \rceil]$, and ℓ is a meaningless padding; then, $w_x^{(1)}(j)$ outputs the k^{th} bit in the binary representation of $\hat{\alpha}_0(\vec{u})$.

Faithful representation. By the faithful representation of Proposition 6.4, the truth-table of $\hat{\alpha}_{d'} \equiv \hat{\alpha}_{d',m}$ is the i^{th} layer of $C'_n(x)$. Thus, $\{\mathtt{OUT}_n\}$ can be implemented by the logspace-uniform AC^0 circuit of size $\mathrm{poly}(d, \log k, \log p) \leq T^{O(\delta)}$ that and implements the weakly systematic property of Enc (see Claim 5.25) for $w_x^{(\bar{d})} = \mathsf{Enc}(\hat{\alpha}_{d',m})$.

Base case. Note that computing $i \mapsto w_x^{(1)}(i)$ reduces to computing $\hat{\alpha}_0$, where the computational costs of the reduction are parsing the input, computing ξ, and printing a single index (all of which can be done in logspace-uniform TC^0 of size polynomial in the input length $\log(T^\kappa) = O(\log T)$). Thus, the base case follows using the logspace-uniform TC^0 circuit from the base case of Proposition 6.4.

Downward self-reducibility. Let us first explain how $\mathtt{DSR}_{n,i}$ is computed, and then bound its complexity. For $i \in \{2, ..., d'\}$, the procedure $\mathtt{DSR}_{n,i}$ gets input $k \in [T^\kappa]$ and acts as follows:

1. It uses the local encoding algorithm from Theorem 5.2 to obtain

   $$D = d \cdot \mathrm{poly}(1/\varepsilon, \log(1/\delta)) = T^{O(\delta)}$$

   locations $q_1, ..., q_D \in [p^{2m}]$ such that $w_x^{(i)}(j)$ depends only on $P_i(q_1), ..., P_i(q_D)$.

2. It uses the downward self-reducibility algorithm from Proposition 6.4 and its oracle access to $w_x^{(i-1)}$ and to S (the specific machine S will be described below) to compute $P_i(q_1), ..., P_i(q_D)$ in parallel.

3. Then, it computes the value of $w_x^{(i)}(j)$ as a function of $P_i(q_1), ..., P_i(q_D)$, using the local encoding algorithm from Theorem 5.2 again.

(Note that the description above uses the fact that Enc is systematic, and in fact uses it twice: Both for $w_x^{(i)} = \mathsf{Enc}(P_i)$ and for $w_x^{(i-1)} = \mathsf{Enc}(P_{i-1})$.)

Recall that the encoding algorithm from Theorem 5.2 is a logspace-uniform TC^0 circuit of size $T^{O(\delta)}$, and that the systematic property of the code uses a logspace-uniform AC^0 circuit of size $T^{O(\delta)}$ (see Claim 5.25). Thus, it is left to describe how to implement the downward self-reducibility in Step (2) above. Recall that the algorithm in Proposition 6.4 is comprised of two parts:

- A machine S that gets as input $2m = O(1/\delta)$ elements of $\mathbb{F}$ and advice φ of length $\mathrm{poly}(T)$, and outputs an element in $\mathbb{F}$ (where φ can be produced from input 1^n in space $O(\log T)$).
- A logspace-uniform circuit of size $h^{c''} \le T^{O(\delta)}$ that uses non-adaptive oracle queries to S and to the preceding polynomial (recall that c'' is the universal constant from Proposition 6.4). Specifically, if the circuit is trying to compute $\hat{\alpha}_{i',0}$ then it gets oracle access to $\hat{\alpha}_{i'-1}$, and if it is trying to compute $\hat{\alpha}_{i',j'}$ then it gets oracle access to $\hat{\alpha}_{i',j'-1}$.

Recall that we are implementing $\mathtt{DSR}_{n,i}$ that tries to compute $w_x^{(i)} = \mathsf{Enc}(P_i)$ with oracle access to $w_x^{(i-1)} = \mathsf{Enc}(P_{i-1})$ (and to S). By our mapping of polynomials $\hat{\alpha}_i$ and $\hat{\alpha}_{i,j}$ to the indexed sequence $P_1, ..., P_{\bar{d}}$, it will always be the case that the preceding polynomial for P_i (as defined above) is P_{i-1}.

By combining the encoding algorithm from Theorem 5.2, the downward self reducibility algorithm from Proposition 6.4 as described above, and the $\mathbf{AC}^0$ circuit from the systematic property of the code, we deduce that the entire procedure $\mathtt{DSR}_{n,i}$ can be computed by a logspace-uniform $\mathbf{TC}^0$ circuit of size $T^{O(\delta)}$ that has oracle access to $w_x^{(i-1)}$ and to S.

Deterministic layer reconstruction. At a high-level, the circuit $\mathtt{REC}_{n,y,i}$ will combine the approximate local list-decoder from Theorem 5.2 for Enc, with the the unique decoder of the Reed-Muller code. To see why, recall that $w_x^{(i)} = \mathsf{Enc}(P_i)$. The decoder for Enc will allow to compute P_i correctly on $1 - 2d^{-c'}$ of the inputs; and the unique decoder for the RM code will use that to compute P_i correctly on all inputs. Details follow.

<u>Approximate local list-decoder of Enc.</u> Let us start by describing a logspace-uniform circuit that implements the decoder from Theorem 5.2. The decoder returns a list of $\mathrm{poly}(1/\varepsilon)$ candidates, and our circuit will test each of the candidates in parallel for agreement with P_i, choosing the best one. Crucially, the logspace algorithm that constructs the circuit will use a sampler to choose fixed randomness (for the decoder, and for the testing of candidates) and will hard-wire them into the circuit.

Claim 6.5.1. *There exists an algorithm A_1 that gets a seed $(y_1, y_2) \in \{0,1\}^{O(\log T)}$, runs in space $O(\log T)$, and prints an oracle $\mathbf{TC}^0$ circuit $\widetilde{C}_{n,(y_1,y_2),i} \colon \mathbb{F}^{2m} \to \mathbb{F}$ of size $T^{O(\delta)}$ that satisfies the following. The circuit first makes non-adaptive preprocessing queries to $w_x^{(i)}$. Now, let $\mathcal{O} \colon [T^\kappa] \to \{0,1\}$ be such that $\Pr_j[\mathcal{O}(j) = w_x^{(i)}(j)] \ge 1/2 + \varepsilon$. Then, with probability at least $1 - \eta/2$ over (y_1, y_2), we have that $\widetilde{C}_{n,(y_1,y_2),i}$ agrees with P_i on $1 - 2d^{-c'}$ of the inputs.*

Proof. Consider the approximate decoder Dec from Theorem 5.2, and let $\ell = O(m \cdot \log p + \log(d/\varepsilon)) = O(\log T)$ be its seed length. Also consider the sampler Γ from Theorem 3.12, instantiated with output length $2m \cdot \log p$, with accuracy $d^{-c'}$, and with confidence $(\eta/4) \cdot \varepsilon^3$; for these parameters, the randomness complexity of Γ is $O(m \cdot \log(p) + \log(d) + \log(1/\varepsilon)) = O(\log T)$ and its sample size is $t = \mathrm{poly}(\log(1/\varepsilon), d) = T^{O(\delta)}$.

The algorithm A_1 gets a seed $y = (y_1, y_2)$ of length $O(\ell)$ representing a seed y_1 for Dec and a seed y_2 for Γ. For each $u \in [L = \tilde{O}(\varepsilon^{-2})]$, it uses Dec with seed y_1 and index u to print a TC^0 circuit $C_{y_1,u}$, which will be a sub-circuit hard-wired into $\widetilde{C}_{n,y,i}$ (see below). Also, the algorithm A_1 computes the outputs $s_1, ..., s_t \in \mathbb{F}^{2m}$ of the sampler with seed y_2. Finally, A_1 prints $\widetilde{C}_{n,(y_1,y_2),i}$ that gets input $\vec{v} \in \mathbb{F}^{2m}$ and performs the following:

1. For each $u \in [L]$ in parallel, let $C_{y_1,u}$ issue its preprocessing queries to $w_x^{(i)}$.
2. For each $u \in [L]$ in parallel, compute $\nu_u = \Pr_{a\in[t]}[C_{y_1,u}(s_a) = w_x^{(i)}(s_a)]$, using the hard-wired points $s_1, ..., s_t$, the oracle access to $w_x^{(i)}$, and the fact that $w_x^{(i)} = \mathsf{Enc}(P_i)$ where Enc is systematic (as in Claim 5.25).
3. Find $u^\star \in [t]$ for which ν_u is maximal, breaking ties arbitrarily.
4. Output $C_{y_1,u^\star}(\vec{v})$.

Let us now bound the size of $\widetilde{C}_{n,(y_1,y_2),i}$. By Theorem 5.2, each of the circuits $C_{y_1,u}$ that Dec outputs is of size

$$s_{\mathsf{Dec}} = \left(\frac{k^\gamma \cdot d \cdot \log(|\mathbb{F}|/\delta)}{\varepsilon}\right)^c < \left(\underbrace{T^{O_{c''}(\gamma/\delta)}}_{k^\gamma=p^{2m\gamma}} \cdot \underbrace{T^{O_{c',c''}(\delta)}}_{d} \cdot \underbrace{T^\delta}_{\varepsilon^{-1}}\right)^{2c} < T^{O_{c,c',c''}(\delta)} ,$$

where $c > 1$ is the universal constant from Theorem 5.2 and we relied on $\gamma = \delta^2$. Thus, $\widetilde{C}_{n,(y_1,y_2),i}$ is of size

$$L \cdot t \cdot \mathrm{poly}(s_{\mathsf{Dec}}) < T^{O_{c,c',c''}(\delta)} < T^{O(\delta)} .$$

Note that $\widetilde{C}_{n,(y_1,y_2),i}$ makes non-adaptive preprocessing queries to $w_x^{(i)}$, both for the preprocessing of the $C_{y_1,u}$-s and to compute $w_x^{(i)}(s_a)$ for all $a \in [t]$. Also note that A_1 uses space $O(\log T)$, by the properties of Dec and of Γ (and since all other computations, such as computing the ν_j-s and comparing them, are in logspace-uniform TC^0).

Now, fix $\mathcal{O}\colon [T^\kappa] \to \{0,1\}$ such that

$$\Pr_j[\mathcal{O}(j) = w_x^{(i)}(j)] \geq 1/2 + \varepsilon.$$

By the properties of Dec, with probability at least $1 - \eta/4$ over y_1, there is $j \in [\varepsilon^{-3}]$ such that

$$\Pr_{\vec{v}}[C_j(\vec{v}) = P_i(\vec{v})] \geq 1 - d^{c'}.$$

By the properties of the sampler and using a union-bound over $j \in [\varepsilon^{-3}]$, with probability at least $1 - \eta/4$ over y_2, for every j we have that

$$\left|\nu_j - \Pr_{\vec{v}}[C_j(\vec{v}) = P_i(\vec{v})]\right| \leq d^{-c'}.$$

Thus, with probability at least $1 - \eta/2$ over (y_1, y_2) it holds that $\Pr_{\vec{v}}[C_{j^*}(\vec{v}) = P_i(\vec{v})] \geq 1 - 2d^{-c'}$, in which case $\widetilde{C}_{n,(y_1,y_2),i}$ agrees with P_i on $1 - 2d^{-c'}$ of the inputs. $\square$

Decoding the original message. Our algorithm will combine A_1 from Claim 6.5.1 with the unique decoder of the Reed-Muller code. For the latter, we will use the following variation on Claim 5.10.

Claim 6.5.2. *There exists an algorithm* $\mathsf{Dec}_{\mathsf{RM}}$ *that gets as input a confidence parameter* $\delta_{\mathsf{RM}} > 0$ *and a seed* $y \in \{0,1\}^{r=O(m\cdot\log(p/\delta_{\mathsf{RM}}))}$*, runs in deterministic space* $O(m \cdot \log(p/\delta_{\mathsf{RM}}))$*, and outputs a (deterministic) oracle* $\mathbf{TC}^0$ *circuit* C_y *of size* $\mathrm{poly}(\Delta \cdot \log(p/\delta_{\mathsf{RM}}))$ *with the following guarantees.*

- *Let* $\hat{\alpha}\colon \mathbb{F}^{2m} \to \mathbb{F}$ *be a polynomial of total degree* Δ*, and let* $w\colon \mathbb{F}^{2m} \to \mathbb{F}$ *be such that* $\Pr_{\vec{v}\in\mathbb{F}^{2m}}[w(\vec{v}) = \hat{\alpha}(\vec{v})] \geq 1 - \frac{1}{100\Delta}$*. Then,*

$$\Pr_y\left[\forall \vec{u} \in \mathbb{F}^{2m}, C_y^w(\vec{u}) = \hat{\alpha}(\vec{u})\right] \geq 1 - \delta_{\mathsf{RM}}.$$

- C_y *queries* w *in at most* $O(\Delta \cdot \log(p/\delta_{\mathsf{RM}}))$ *locations, non-adaptively.*

Proof sketch. The proof is identical to that of Claim 5.10, just with different parameters. Let Γ' be the sampler from Theorem 3.11, instantiated with accuracy $1/200$ and confidence $\delta_{\mathsf{RM}}/p^{2m}$ and output in $\{0,1\}^{2m\log p} \equiv \mathbb{F}^{2m}$ (i.e., output length $2m\cdot\log(p)$); note that the sample size is $t = O(\log(p^{2m}/\delta_{\mathsf{RM}})) = O(m \cdot \log(p/\delta_{\mathsf{RM}})$, and that indeed $r \leq O(m \cdot \log(p/\delta_{\mathsf{RM}}))$. The algorithm $\mathsf{Dec}_{\mathsf{RM}}$ gets a seed y and computes t vectors $\vec{s}_{y,i} = \Gamma'(y,i) \in \mathbb{F}^{2m}$, for $i = 1,...,t$. It prints circuit C_y that interpolates, for each $i \in [t]$, the degree-Δ univariate $q_{y,i}\colon \mathbb{F} \to \mathbb{F}$ obtained by restricting w to the line $\{\vec{u} + a \cdot \vec{s}_{y,i}\}_{a\in\mathbb{F}}$, lets $v_{y,i} = q_{y,i}(\vec{u})$, and outputs the most common element in the list $V_y = \{v_{y,i}\}_{i\in[t]}$ (breaking ties arbitrarily). The interpolation of $q_{y,i}$ is done by examining the first $(\Delta - 1)$ points of $\{\vec{u} + a \cdot \vec{s}_{y,i}\}_{a\in\mathbb{F}}$.

Recall that a random choice of $\vec{s} \in \mathbb{F}^{2m}$ yields a line through $\vec{u}$ such that the restriction of w to the first $\Delta - 1$ points on the line agrees with $\hat{\alpha}$ on $\vec{u}$, with probability 0.99. By the properties of Γ', for every fixed $\vec{u}$, for all but $\delta_{\mathsf{RM}}/p^{2m}$ of the choices of y it holds that $C_y(\vec{u}) = \hat{\alpha}(\vec{u})$. The correctness follows by a union bound over $\vec{u} \in \mathbb{F}^{2m}$.

As for the complexity, note that the sampler is computable in space linear in r, and thus $\mathsf{Dec}_{\mathsf{RM}}$ can compute the $t \cdot \Delta$ locations on which C_y queries w in space $O(r + \log t)$. Since interpolating a univariate of degree Δ and taking the most common element in a list of size t can be done by a logspace-uniform $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(t, \Delta, \log p)$, the algorithm $\mathsf{Dec}_{\mathsf{RM}}$ runs in space $O(m \cdot \log(p/\delta_{\mathsf{RM}}))$. ∎

Now, let us describe an algorithm A_2 that gets a seed of the form $\vec{y} = (y_1, y_2, y_3)$ and prints a circuit $C_{n,\vec{y},i}$. The algorithm A_2 uses the following components:

1. The algorithm A_1 from Claim 6.5.1, instantiated with seed (y_1, y_2), which prints an oracle $\mathbf{TC}^0$ circuit $\widetilde{C}_{n,(y_1,y_2),i}$.

2. The algorithm $\mathsf{Dec}_{\mathsf{RM}}$ from Claim 6.5.2, instantiated with confidence parameter $\delta_{\mathsf{RM}} = \eta/2$ and seed y_3, which prints an oracle $\mathbf{TC}^0$ circuit C_{y_3}.

The algorithm A_2 prints a circuit $C_{n,\vec{y},i}$ that has $\widetilde{C}_{n,(y_1,y_2),i}$ and C_{y_3} hard-wired, and acts as follows: For pre-processing, it runs the pre-processing of $\widetilde{C}_{n,(y_1,y_2),i}$, providing it access to $w_x^{(i)}$; and when given input $j \in [T^\kappa]$, it outputs

$$C_{y_3}^{\widetilde{C}^{\mathcal{O}}_{n,(y_1,y_2),i}}(j) \ .$$

In words, after pre-processing and when receiving input j, the circuit simulates C_{y_3} on j; whenever C_{y_3} makes an oracle query q, the circuit resolves it simulating $\widetilde{C}_{n,(y_1,y_2),i}(q)$; whenever the latter circuit makes an oracle query q', it is resolved by the oracle $\mathcal{O}$.

Note that the seed length for A_2 is $O(m \cdot \log p) = O(\log T)$, and that its space complexity is $O(\log T)$ (using efficient space-bounded composition with A_1 and with $\mathsf{Dec}_{\mathsf{RM}}$). The circuit $C_{n,\vec{y},i}$ that it prints is of size at most

$$|C_{y_3}| \cdot \mathrm{poly}(|\widetilde{C}_{n,(y_1,y_2),i}|) = \mathrm{poly}\big(\Delta, \log(p/\delta_{\mathsf{RM}}), T^{O(\delta)}\big) \le T^{O(\delta)} \ .$$

Indeed, this circuit makes queries to $w_i^{(x)}$ in the preprocessing step (and these are independent of any input), and makes queries to $\mathcal{O}$ in the computation step.

Finally, fix $\mathcal{O}$ such that $\Pr_j[\mathcal{O}(j) = w_x^{(i)}(j)] \ge 1/2 + \varepsilon$. By Claim 6.5.1, with probability at least $1 - \eta/2$ over (y_1, y_2) we have

$$\Pr_{\vec{v} \in \mathbb{F}^{2m}}\left[C^{\mathcal{O}}_{n,(y_1,y_2),i}(\vec{v}) = P_i(\vec{v})\right] \ge 1 - 2d^{-c'} > 1 - \frac{1}{100\Delta} \ . \tag{6.2}$$

Conditioned on the event above, by Claim 6.5.2, with probability at least $1 - \eta/2$ over y_3 it holds that $C^{O}_{n,\vec{y},i}$ correctly computes P_i on all inputs.

<u>Decoding the final codeword.</u> Finally, we transform $C^{\mathcal{O}}_{n,\vec{y},i}$, which computes P_i, into a circuit that computes $w_x^{(i)} = \mathsf{Enc}(P_i)$. Specifically, the final algorithm A gets seed $\vec{y}$ (i.e., exactly the same as A_2), and prints a circuit $\mathtt{REC}_{n,\vec{y},i}$ that, on input $j \in [T^\kappa]$, uses the logspace-uniform $\mathbf{TC}^0$ circuit from Theorem 5.2 to compute $D \le T^{O(\delta)}$ locations in P_i corresponding to the j^{th} output; then uses $C_{n,\vec{y},i}$ to compute P_i on these locations; and finally uses the circuit for local encoding from Theorem 5.2 to compute the j^{th} output of $\mathsf{Enc}(P_i)$. Note that the complexity of A is dominated by A_2, and the complexity of $\mathtt{REC}_{n,\vec{y},i}$ is dominated by $C_{n,\vec{y},i}$

Computing the $w_x^{(i)}$-s in small space. The last thing to prove is that there is an $O(\log T)$-space algorithm mapping (x, i, j) to $w_x^{(i)}(j)$. To see this, consider the naive combination of the sequence of reductions, from computing $w_x^{(i)}$ to computing $w_x^{(i-1)}$, all the way down to computing $w_x^{(1)}$. This sequence can be modeled as a tree of depth $i \le \bar{d}$ and fan-in $T^{O(\delta)}$, where the function at each node is computable in space $O(\log T)$ (see justification below). Thus, by running the standard DFS-style bounded-space simulation on the depth-$\bar{d}$ tree, we can compute $w_x^{(i)}(j)$ in space $O(\bar{d} \cdot \log T) = O(\log T)$.

The only missing piece is to see that each node is computable in space $O(\log T)$. To see this, note that each node in layer $j \in [i]$ is computable by $\mathtt{DSR}_{n,j}$ with oracle access to S. Recall that $\mathtt{DSR}_{n,j}$ is a logspace-uniform TC^0 circuit of size $T^{O(\delta)}$, and that S is computable in space $O(\delta \cdot \log T)$ when given advice φ that can be generated in space $O(\log T)$. Hence, we can compute S by giving it virtual access to φ, and using the DFS-style simulation of the TC_0 circuit, we can compute $\mathtt{DSR}^S_{n,j}$ in space $O(\log T)$. ■

6.1.2 The reconstructive targeted somewhere-PRG

We will need a modified version of the classical Nisan-Wigderson [NW94] generator, when its reconstruction argument is uniform as in [IW98]. In the modified version, the uniform reconstruction argument is a *space-efficient* probabilistic Turing machine that prints a circuit computing the hard function; and there are a few additional non-standard points:

1. Recall that the NW reconstruction argument succeeds with relatively low probability (roughly $\approx 1/m$ where m is the length of the pseudorandom output string), and needs to be repeated $O(m)$ times to achieve high success probability. In the modified version, the task of checking which of the $O(m)$ attempts was successful is delegated to the printed circuit, rather than to the Turing machine that prints it.

2. The machine printing the circuit only uses $O(\log m)$ random coins, and the circuit is deterministic. This is achieved by using the consistency test given in Theorem 4.2 (to avoid randomly choosing a $\Theta(m)$-bit string as in the standard reconstruction of NW), and by using randomness-efficient samplers (to execute the many attempts of NW reconstruction, and check each of them for success, in a randomness-efficient way).

Let us formally state this version and prove it.

Theorem 6.6 (the NW PRG)**.** *There exist a universal constant c_{NW} such that for any two constants $\eta, \delta_{\mathsf{NW}} > 0$ there are two deterministic algorithms G^{NW} and R^{NW} satisfying the following.*

1. ***Generator.*** *On input 1^n and oracle access to a string $f \in \{0,1\}^n$, the algorithm G^{NW} runs in space $c_{\mathsf{NW}} \cdot \log n$ and prints a list of at most $n^{c_{\mathsf{NW}}}$ strings in $\{0,1\}^m$, where $m = n^{1/c_{\mathsf{NW}}}$.*

2. ***Reconstruction.*** *On input $(1^n, w)$ and a random seed y_{NW} of length $c_{\mathsf{NW}} \cdot \log(nw)$, the algorithm R^{NW} runs in space $c_{\mathsf{NW}} \cdot \log(nw)$ and prints a non-adaptive oracle TC^0 circuit $C^{\mathsf{NW}}_{y_{\mathsf{NW}}}$ of size $(m \cdot w)^{c_{\mathsf{NW}}}$ that satisfies the following.*

   For every ROBP *D of length m and width w that δ_{NW}-distinguishes $(G^{\mathsf{NW}})^f(1^n)$ from uniform, with probability at least $1 - \eta$ over y_{NW} the following holds.*

   The circuit $C^{\mathsf{NW}}_{y_{\mathsf{NW}}}$ has a preprocessing step in which it queries f. Then, in the computation step it satisfies

   $$\Pr_{i \in [n]}\left[(C^{\mathsf{NW}}_{y_{\mathsf{NW}}})^{\widetilde{D}}(i) = f_i\right] \geq \frac{1}{2} + \frac{\delta_{\mathsf{NW}}}{8m},$$

where $\widetilde{D}$ is a function that gets as input (r, a, b) and outputs $D_{a,b}(r)$.[36]

Proof. The algorithm G^{NW} constructs a combinatorial design $S_1, ..., S_m \subseteq [d]$ with sets of size $|S_i| = \log n$ and with pairwise intersections $|S_i \cap S_j| \leq 10 \log m$ for distinct $i, j \in [m]$ and with $d = O(\log n)$. Recall that, by Theorem 3.16, this can be done in space $O(\log n)$. For every $s \in \{0,1\}^d$, the s^{th} output string in the list is $(f_{z\restriction_{S_1}}, ..., f_{z\restriction_{S_m}}) \in \{0,1\}^m$.

Description of R^{NW}. Let $r = \lceil d - \log n + \log m + \log w + 1 \rceil = O(\log(nw))$. Consider the sampler

$$\Gamma \colon \{0,1\}^{O(r)} \times [t] \to \{0,1\}^r$$

from Theorem 3.12, instantiated with confidence $\eta/2$ and accuracy $\delta_{\mathsf{NW}}/(8m^2w)$, where $t = \text{poly}(m, w)$. Note that Γ is computable in space $O(\log(nw))$.

For each sample $\ell \in \{0,1\}^r$ in the output of Γ, the algorithm R^{NW} interprets $\ell = (z, i, j, b) \in \{0,1\}^{[d] \setminus S_i} \times [m] \times [w] \times \{0,1\}$. Consider a circuit C_ℓ that has ℓ hard-wired, gets input $\alpha \in [n]$, completes z to $z_\alpha \in \{0,1\}^d$ by placing α in the locations corresponding to S_i, and outputs

$$D_{i,j}\left(f_{z_\alpha\restriction_{S_{m-i+1}}}, ..., f_{z_\alpha\restriction_{S_m}}\right) \oplus b.$$

To be able to perform this computation on input $\alpha \in [n]$, at the preprocessing step, the circuit C_ℓ queries f at locations

$$\left\{z_{\alpha'}\restriction_{S_j}\right\}_{j \in \{m-i+1,...,m\}, \alpha' \in [n]}.$$

Note that the number of queries is at most $m \cdot 2^{10 \cdot \log(m)}$, since the design ensures that $|S_j \cap S_i| \leq 10 \cdot \log m$ for every $j \neq i$. Also, the queries are independent of any input α, and can thus be made at preprocessing.

Now, R^{NW} prints a circuit $C = C^{\mathsf{NW}}$ that has each C_ℓ as a sub-circuit. To do so, consider another instantiation Γ' of the sampler from Theorem 3.12, this time with accuracy $\delta_{\mathsf{NW}}/4m$ and confidence $\eta/2t$ and parameters

$$\Gamma' \colon \{0,1\}^{O(\log(n \cdot w))} \times [t'] \to \{0,1\}^{\log(n)},$$

where $t' = \text{poly}(m/\delta_{\mathsf{NW}})$. Then, R^{NW} uses Γ' to obtain samples $\alpha_1, ..., \alpha_{t'} \in \{0,1\}^{\log(n)}$. At the pre-processing step, the circuit C queries f at locations $\alpha_1, ..., \alpha_{t'}$; lets each C_ℓ query f; and computes $\ell^\star$ that maximizes the value $\nu_\ell = \Pr_{k \in [t']}[C_\ell(\alpha_k) = f_{\alpha_k}]$. When receiving input $\alpha \in [n]$, the circuit C outputs $C_{\ell^\star}(\alpha)$.

[36] Recall, from Definition 4.1, that $D_{a,b}$ is the ROBP that executes the sub-procedure of D starting from vertex $b \in [w]$ at layer $a \in [m]$.

Analysis of R^{NW}. Note that the seed length for R^{NW} is $O(\log(nw))$. Also, each C_ℓ is logspace-uniform,[37] and thus C is also logspace-uniform. The space complexity of R^{NW} is thus dominated by the space complexity of computing the samplers, which is $O(\log(n \cdot w))$. The circuit C that it outputs is of size

$$\mathrm{poly}(t, m, t') = \mathrm{poly}(m, w),$$

and indeed C queries f in the preprocessing step and $\widetilde{D}$ (as we defined in the theorem statement) in the computation step.

By Theorem 4.2 and the hypothesis that D is a δ_{NW}-distinguisher for G^{NW}, there exist (i, j, b) such that

$$\Pr_{y \in \{0,1\}^d}\left[D_{i,j}\left(f_{y\restriction_{S_{m-i+1}}}, \ldots, f_{y\restriction_{S_m}}\right) \oplus b = f_{y}\restriction_{S_i}\right] \geq \frac{1}{2} + \frac{\delta_{\mathsf{NW}}}{m} ,$$

or equivalently

$$\mathop{\mathbb{E}}_{z \in \{0,1\}^{[d] \setminus S_i}}\left[\Pr_{\alpha \in [n]}\left[D_{i,j}\left(f_{z_\alpha\restriction_{S_{m-i+1}}}, \ldots, f_{z_\alpha\restriction_{S_m}}\right) \oplus b = f_\alpha\right] - \frac{1}{2}\right] \geq \frac{\delta_{\mathsf{NW}}}{m} .$$

Thus, with probability at least $\delta_{\mathsf{NW}}/2m$ over $z \in \{0,1\}^{[d] \setminus S_i}$ it holds that

$$\Pr_{\alpha \in [n]}\left[D_{i,j}\left(f_{z_\alpha\restriction_{S_{m-i+1}}}, \ldots, f_{z_\alpha\restriction_{S_m}}\right) \oplus b = f_\alpha\ \right] \geq \frac{1}{2} + \frac{\delta_{\mathsf{NW}}}{2m} . \tag{6.3}$$

It follows that, over uniform choices of (z, i, j, b), Equation (6.3) holds with probability at least $(\delta_{\mathsf{NW}}/2m) \cdot (1/m) \cdot (1/w) \cdot (1/2) = \frac{\delta_{\mathsf{NW}}}{4m^2 \cdot w}$. By our choice of parameters for Γ, with probability at least $1 - \eta/2$ over the seed for R^{NW}, there exists $\ell = (z, i, j, b)$ in the output sample of Γ such that Equation (6.3) holds. For such an ℓ, we have that $\Pr_{\alpha \in [n]}[C_\ell(\alpha) = f_\alpha] \geq 1/2 + \delta_{\mathsf{NW}}/2m$.

Also note that with probability at least $1 - \eta/2$ over a choice of seed for R^{NW}, for each ℓ it holds that $\left|\nu_\ell - \Pr_\alpha[C_\ell(\alpha) = f_\alpha]\right| \leq \delta_{\mathsf{NW}}/4m$. Whenever this happens, we have that

$$\Pr_{\alpha \in [n]}[C(\alpha) = f_\alpha] = \Pr_{\alpha \in [n]}[C_{\ell^\star}(\alpha) = f_\alpha] \geq \frac{1}{2} + \frac{\delta_{\mathsf{NW}}}{4m} ,$$

as we wanted. ∎

We are now ready to present the reconstructive targeted somewhere-PRG, which will be based on a hard function in logspace-uniform $\mathbf{TC}^0$. For every fixed input x, the targeted somewhere-PRG outputs a sequence of lists. If an ROBP D distinguishes each of the lists in the sequence from uniform, then we can compute the hard function on x in $(\mathbf{TC}^0)^D$ of bounded size, where the reconstruction argument (that prints this oracle $\mathbf{TC}^0$ circuit) uses a random seed of only logarithmic length.

[37] To see this, recall that almost all of C_ℓ is a static "lookup table" containing $m \cdot 2^{10 \cdot \log(m)}$ values. The actual functionality of C_ℓ consists of placing its input α in locations of z to get z_α; querying the lookup table at inputs that are obtained by examining fixed location-sets in z, to obtain a string q; and invoking the oracle on q.

Theorem 6.7 (a reconstructive targeted somewhere-PRG with log-seed logspace-uniform TC^0 reconstruction)**.** *There is a universal constant $c > 1$ such that for every $\alpha, \beta, \delta \in (0,1)$ and $d \in \mathbb{N}$ the following holds. Let $T, r\colon \mathbb{N} \to \mathbb{N}$ such that $T(n) \geq n$ is computable in space $O(\log T)$, and let $m(n) = T(n)^{\delta/c}$.*

Let $f\colon \{0,1\}^n \to \{0,1\}^{r(n)}$ be computable by a family of logspace-uniform TC^0 circuits of depth d and size T. Then, there exist deterministic algorithms G_f, R_f, and $\mathcal{O}_f$, that satisfy the following.

1. ***Generator.*** *On input $x \in \{0,1\}^n$, the algorithm G_f runs in space $O(\log T)$ and prints $\bar{d} = c \cdot (d/\delta^2)$ lists $G_f(x)_1, ..., G_f(x)_{\bar{d}}$, where each list contains $\mathrm{poly}(T)$ strings in $\{0,1\}^m$.*

2. ***Reconstruction.*** *On input 1^n and a seed $y \in \{0,1\}^{O(\log T)}$ and a description of a machine M that runs in space $\log(m)$,*[38] *the algorithm R_f runs in space $O(\log T)$ and prints an oracle TC^0 circuit C_y of depth $c \cdot (d/\delta^2)$ and size $(n \cdot m)^c$ such that for every fixed input $x \in \{0,1\}^n$ the following holds.*

   Assume that for every $i \in [\bar{d}]$ it holds that $M(x, \cdot)$ β-distinguishes the uniform distribution over $G_f(x)_i$ from U_m. Then, with probability at least $1 - \alpha$ over y it holds that $C_y^{\mathcal{O}}(x)$ prints a description of an oracle TC^0 circuit $F_{x,y}$ such that the truth-table of $F_{x,y}^{\mathcal{O}}$ is $f(x)$, where queries to $\mathcal{O}$ are of length at most $n + 2m + \mathrm{polylog}(T)$.

3. ***Oracle.*** *For any $n \in \mathbb{N}$, the machine $\mathcal{O}$ gets inputs of length $n + 2m + \mathrm{polylog}(T)$ and runs in space $c \cdot \delta \cdot \log T$.*

Furthermore, for every x there is a sequence of $\bar{d}$ strings $w_x^{(1)}, ..., w_x^{(\bar{d})}$ of length $\mathrm{poly}(T)$ such that $w_x^{(1)}$ can be printed in space $O(\log T)$ given access to x, and for all $i \in \{2, ..., \bar{d}\}$, the string $w_x^{(i)}$ can be printed in space $O(\log T)$ with oracle access to $w_x^{(i-1)}$, and the following holds:

- *When we give R_f an additional input $i \in [\bar{d}]$ (i.e., in addition to 1^n and the seed y), it outputs a circuit $C_{y,i}$.*
- *We say that R_f* is successful for x with i *if with probability at least $1 - \alpha \cdot (i/\bar{d})$ over y it outputs $C_{y,i}$ such that $F_{x,y,i} = C_{y,i}^{\mathcal{O}}(x)$ is a TC^0 circuit satisfying $\mathsf{tt}\big(F_{x,y,i}^{\mathcal{O}}\big) = w_x^{(i)}$. Then,*
  - *R_f is successful for any x with $i = 1$.*
  - *Let $i \in \{2, ..., \bar{d}\}$, and assume that R_f is successful for x with $i - 1$ and that $M(x, \cdot)$ is β-distinguisher for $G_f(x)_i$. Then, R_f is successful for x with i.*
  - *If $\Pr_y\big[\mathsf{tt}\big(F_{x,y}^{\mathcal{O}}\big) = f(x)\big] < 1 - \alpha$, then R_f is* not *successful for x with $\bar{d}$.*

Proof. We use Proposition 6.5 with the parameter δ and with a sufficiently small constant $\eta < \alpha/2\bar{d}$, where $\bar{d} = O(d/\delta^2)$. We also use Theorem 6.6, with $\delta_{\mathsf{NW}} = \beta$, with the constant η,

[38] Formally, for every machine M with a fixed description size, we analyze the behavior of R_f when n is sufficiently large.

with input length T^κ, and with output length $T^{\kappa/c_{\mathsf{NW}}} \geq m$ (using the fact that $m = T^{\delta/c}$ for a sufficiently large universal $c > 1$. Let G^{NW} and R^{NW} be the algorithms from Theorem 6.6.

Fix an input $x \in \{0,1\}^n$, and let $w_x^{(1)}, ..., w_x^{(\bar{d})} : [T^\kappa] \to \{0,1\}$ be the sequence of functions from Proposition 6.5. For each $i \in [\bar{d}]$, the generator $G_f(x)$ prints the list of strings given by $(G^{\mathsf{NW}})^{w_x^{(i)}}(1^T)$, where each string is truncated to be of length m. Note that G_f runs in space $O(\log \bar{d} + \log T + \log m) = O(\log T)$, because it prints $\bar{d}$ lists, and the space complexity of G_f and of computing $w_x^{(i)}$ is $O(\log T)$ (see the "Lastly" part of Proposition 6.5), and prints $\bar{d}$ lists, each with $T^{c_{\mathsf{NW}} \cdot \kappa} = \mathrm{poly}(T)$ strings.

Oracle. We define a machine $\mathcal{O}$ that, for any $n \in \mathbb{N}$, gets two types of inputs:

1. The first type of input has form $(\langle M \rangle, x, r, a, b)$, where M is a description of a probabilistic Turing machine (the description is of length at most $\log(|x|)$). The machine $\mathcal{O}$ runs M on input x from the initial state b, for $m - a$ steps and using at most $\log(m)$ space, with the $(m-a)^{\text{th}}$ prefix of r as random coins, and prints M's output. (If M exceeds the time or space bounds, $\mathcal{O}$ halts and prints a default output.)

   For every fixed $x \in \{0,1\}^n$, let $D_x(r) = M(x,r)$, and observe that D_x is an ROBP. Also observe that $\mathcal{O}(\langle M \rangle, x, r, a, b) = (D_x)_{a,b}(r)$, where the latter ROBP is obtained from D_x by starting from vertex b at layer a (recall Definition 4.1).

2. The second type of input specifies $O(1/\delta)$ elements of $\mathbb{F}$ as well as a string $\varphi \in \{0,1\}^{\mathrm{polylog}(T)}$ additional bits. The machine $\mathcal{O}$ runs the machine S from the downward self-reducibility of Proposition 6.5, providing it with φ as advice.

Overall, for any $n \in \mathbb{N}$, the queries to $\mathcal{O}$ are of length at most

$$\max\{n + m + 2\log(m) + \log(n), \mathrm{polylog}(T)\} < n + 2m + \mathrm{polylog}(T)$$

and $\mathcal{O}$ runs in space $\max\{O(\delta) \cdot \log(T), \log(m)\} < c \cdot \delta \cdot \log(T)$, where we assume again that the universal constant $c > 1$ is sufficiently large.

Reconstruction. The machine R_f uses its seed y to sample $2\bar{d}$ seeds of length $O(\log T)$, denoted $y_{1,1}, y_{1,2}, ..., y_{\bar{d},1}, y_{\bar{d},2}$. We will first describe the circuit C_y that R_f prints and the oracles, and then verify that they meet the claimed specifications.

The circuit C_y works in $\bar{d}$ steps. For $i \in [\bar{d}]$, the goal of the i^{th} step is to compute a description of a $\mathbf{TC}^0$ circuit $C_{y,i}$ such that the truth-table of $C_{y,i}^{\widetilde{D}}$ is $w_x^{(i)}$, where $\widetilde{D}(r,a,b)$ runs M on input x from the initial state b, for $m - a$ steps.

1. For the base case $i = 1$, the circuit $C_{y,1}$ is simply the circuit $\mathtt{BASE}_n$ from Proposition 6.5.

2. For $i \in \{2, ..., \bar{d}\}$, the circuit C_y already computed a description of $C_{y,i-1}$ such that $\mathrm{tt}(C_{y,i-1}^{\widetilde{D}}) = w_x^{(i-1)}$. Note that any query to $w_x^{(i)}$ can be answered by using the downward self-reducibility algorithm $\mathtt{DSR}_{n,i}$ from Proposition 6.5, as follows. The circuit answers the queries of $\mathtt{DSR}_{n,i}$ to $w_x^{(i-1)}$ using $C_{y,i-1}$ and the oracle access to $\mathcal{O}$

(i.e., whenever $C_{y,i-1}$ queries $\widetilde{D}$ at (r,a,b) we use the first type of query to $\mathcal{O}$, i.e. $(\langle M\rangle, x, r, a, b)$); and it answers the queries to S by the oracle access to $\mathcal{O}$ (i.e., whenever $C_{y,i-1}$ queries S at $\sigma_1, ..., \sigma_{O(1/\delta)}$ we use the second type of queries to $\mathcal{O}$, i.e. send the σ_i's along with an advice φ for S that is hard-wired into C_y).

Now, C_y uses the circuit C^{NW} given by R^{NW} with seed $y_{i,1}$, and runs its preprocessing step, while answering its queries to $w_x^{(i)}$ as explained above. It then uses layer reconstruction $\mathtt{REC}_{n,y_{i,2},i}$ from Proposition 6.5, and runs its preprocessing step, while answering its queries to $w_x^{(i)}$ and to S in the same way.

The circuit $C_{y,i}\colon [T^\kappa] \to \{0,1\}$ is defined as

$$C^{\mathcal{O}}_{y,i}(j) = \mathtt{REC}^{(C^{\mathsf{NW}})^{\mathcal{O}}}_{n,y_{i,2},i}(j).$$

3. Finally, the circuit C_y simulates the preprocessing step of $C_{y,\bar{d}}$ (answering queries as in each step above), and prints the description of a circuit $F_{x,y}$ that implements $\mathtt{OUT}_n$ from Proposition 6.5, while resolving oracle queries of $\mathtt{OUT}_n$ with $C_{y,\bar{d}}$.

Correctness. Fix a space-$\log m$ machine M, and let x be an input such that $M(x,\cdot)$ β-distinguishes $G_f(x)_i$ from uniform for all $i \in [\bar{d}]$. Let $D = D_x$ be the ROBP $D(r) = M(x,r)$, and note that D is a β-distinguisher for $(G^{\mathsf{NW}})^{w_x^{(i)}}(1^T)$, for every $i \in [\bar{d}]$. Also observe that for every $i \in [\bar{d}]$ all of the queries of the circuit C^{NW} (given by R^{NW}) are answered by $\widetilde{D}(r,a,b) = D_{a,b}(r)$, as is required for the reconstruction in Theorem 6.6.

Now, recall that the seed of R_f specifies $\bar{d}$ seeds for R^{NW} and $\bar{d}$ seeds for the layer reconstruction (from Proposition 6.5). We prove by induction on $i \in [\bar{d}]$ that, with probability at least $1 - 2i\cdot\eta$ over choice of seed for R_f, it holds that $C_i^{\mathcal{O}} \equiv w_x^{(i)}$.

The base case follows from the base case of Proposition 6.5. For $i \in \{2, ..., \bar{d}\}$, the induction hypothesis implies that the preprocessing step for C^{NW} and for $\mathtt{REC}_{n,y_{i,2},i}$ will be executed correctly. Then, with probability at least $1-\eta$ over $y_{i,1}$ it holds that $C_i^{\mathcal{O}} = (C^{\mathsf{NW}})^{\mathcal{O}}$ computes $w_x^{(i)}$ correctly on $1/2 + \beta/8m$ of the inputs. In this case, with probability at least $1-\eta$ over $y_{i,2}$ it holds that $\mathtt{REC}^{(C^{\mathsf{NW}})^{\mathcal{O}}}_{n,y_2,i}$ computes $w_x^{(i)}$ correctly on all inputs, where we used the fact that $\beta/8m > T^{-\delta/c_1}$ where c_1 is the universal constant from Proposition 6.5.

By our choice of $\eta < \alpha/2\bar{d}$, with probability at least $1-\alpha$ we have that $C_{\bar{d}}(j) = w_x^{(\bar{d})}(j)$ for all $j \in [T^\kappa]$. In this case, by the properties of $\mathtt{OUT}_n$ we have that $C_y(x) = C_n(x)$.

Complexity of R_f and of C_y. Note that the depth of C_y is at most $O(\bar{d}) = O(d/\delta^2)$. To bound its size, let $c_1 = c_{\mathsf{NW}} > 1$ be the universal constant from Theorem 6.6, and let $c_2 > 1$ be the universal constant from Proposition 6.5. Recall that the size of the circuit that R^{NW} prints is at most $(m\cdot w)^{c_1} < m^{2c_1}$, where we relied on the fact that the ROBP $D_x(r) = M(x,r)$ has width m. Also, the size of the circuit for downward self-reducibility from Proposition 6.5 is at most $T^{c_2\cdot\delta}$, and the size of the circuit for the base case is at most $(n\cdot T^\delta)^{c_2}$. Finally, at each step i the circuit C_y will simulate the circuit $C_{y,i-1}$, which is of size at most $T^{c_2\cdot\delta}$. Thus, the total size of C_y is less than

$$\bar{d}\cdot\left((n\cdot T^\delta)^c + m^c\right) < (n\cdot m)^{c^2},$$

where $c = c(c_1, c_2) > 1$ is a sufficiently large universal constant. Also, since all the sub-circuits in C_y are logspace-uniform TC^0 circuits, the space complexity of R_f is at most $O(\log T)$.

Recall that queries to $\mathcal{O}$ require either a description of M (if they are of the first type) or a string φ of size $\mathrm{polylog}(T)$ (if they are of the second type). The algorithm R_f hard-wires the description of M and the string φ into the description of C_y; it can do so because $\langle M \rangle$ is given to R_f as input, and because φ is computable in space $O(\log T)$.

This accounts for the oracle queries that C_y makes to $\mathcal{O}$, but we did not yet account for the queries made in the base case. Specifically, recall that $\mathtt{BASE}_n$ from Proposition 6.5 makes queries to a space-$O(\log T)$ machine B. However, since these queries are non-adaptive and do not depend on x, the machine R_f can compute the answers of B by itself when constructing C_y, and hard-wire them.

The "furthermore" part. The "furthermore" statement follows almost immediately from the same proof, with the strings $w_x^{(1)}, ..., w_x^{(\bar{d})}$ defined above. By the base case of Proposition 6.5, $w_x^{(1)}$ can be printed (given input x) in space $O(\log T)$; and by the downward self-reducibility, $w_x^{(i)}$ can be printed in space $O(\log T)$ with oracle access to $w_x^{(i-1)}$.

Now, when R_f gets an additional input $i \in [\bar{d}]$, it prints a circuit $C'_{y,i}$ that acts as follows: Instead of carrying out the reconstruction for $\bar{d}$ steps, the circuit carries out the reconstruction for only i steps to obtain a description of $C_{y,i}$ (as defined above), simulates the preprocessing of $C_{y,i}$, and prints $F_{x,y,i}$ that is the description of $C_{y,i}$ after preprocessing.

For $i = 1$, this is the base case circuit, so R_f always outputs a correct circuit (i.e., regardless of its seed y).[39] The proof above shows that for every $i \in \{2, ..., \bar{d}\}$, conditioned on R_f outputting a correct circuit $C_{y,i-1}$ and on $M(x, \cdot)$ being a β-distinguisher for $G(x)_i = (G^{\mathsf{NW}})^{w_x^{(i)}}(1^T)$, with probability at least $1 - 2\eta > 1 - \alpha/\bar{d}$ it outputs a correct circuit $C_{y,i}$. In particular, when $M(x, \cdot)$ is a β-distinguisher for $G(x)_i$, we have that

$$\begin{aligned}
&\Pr[R_f(1^n, y, i-1) \text{ prints the correct circuit}] \geq 1 - \alpha \cdot ((i-1)/\bar{d}) \\
&\Longrightarrow \Pr[R_f(1^n, y, i) \text{ prints the correct circuit}] \geq 1 - \alpha \cdot (i/\bar{d}) \; .
\end{aligned}$$

Lastly, if R_f is successful for x with $\bar{d}$, then with probability at least $1 - \alpha$ it holds that $C^{\mathcal{O}}_{y,\bar{d}}(j) = F^{\mathcal{O}}_{x,y,\bar{d}}(j) = w_x^{(\bar{d})}(j)$ for all $j \in [T^\kappa]$. But in this case $F_{x,y}$ defined above (i.e., that implements $\mathtt{OUT}_n$ and resolves its oracle queries with $C_{y,\bar{d}}$) computes the output of $C'_n(x)$, which is $f(x)$. This contradicts the assumption that $\Pr_y[\mathsf{tt}(F^{\mathcal{O}}_{x,y}) = f(x)] < 1 - \alpha$. ■

6.2 Proof of the Main Result

We first prove a weaker version of Theorem 1, in which we assume that every log-spaceadvice-uniform TC^0 circuit family fails to compute the hard function on 99% of its inputs (rather than on 1%). That is:

[39] We say that a circuit $C_{y,i}$ is correct if $C^{\mathcal{O}}_{y,i}(j) = w_x^{(i)}(j)$ for all $j \in [T^\kappa]$.

Theorem 6.8. *Assume that for every $c \in \mathbb{N}$ there are constants $k > c$ and $d \in \mathbb{N}$ and a function $f\colon \{0,1\}^n \to \{0,1\}^{r(n)}$, for some $r\colon \mathbb{N} \to \mathbb{N}$, that satisfies the following:*

1. ***Upper bound.*** *f is computable in logspace-uniform TC^0 of depth d and size n^k.*
2. ***Lower bound.*** *For every log-spaceadvice-uniform $(\mathsf{TC}^0)^{\mathsf{DSPACE}[c\cdot\log(n)]}$ circuit family $\{C_n\}$ of size n^c and depth $c \cdot d \cdot k^2$, and every sufficiently large $n \in \mathbb{N}$, we have*

$$\Pr_{x\in\{0,1\}^n}[C_n(x) = f(x)] \le \varepsilon/2 \; .$$

Then, $\mathsf{BPL} \subseteq \cap_{\varepsilon>0}\mathsf{avg}_\varepsilon\mathsf{L}$.

We stress that in the hypothesized lower bound, the depth of the TC^0 circuit depends on the hard function f (i.e., on the size and depth of the circuit computing f), but its size n^c does not. Also, in the notation describing the oracle $\mathsf{DSPACE}[c \cdot \log(n)]$ machine, our proof shows that the space complexity of the oracle is $c\cdot\log(n)$ where n is the input length to C_n (which is smaller than $c \cdot \log(|q|)$ where q is the query to the oracle).

Proof of Theorem 6.8. Let $L \in \mathsf{BPL}$, decided by a probabilistic machine M using space $c_0 \cdot \log(n)$, and let $\varepsilon > 0$. For a large constant $c = c(c_0) > c_0$ that will be determined in a moment, let $k, d \in \mathbb{N}$ be the corresponding constants from our hypothesis.

We instantiate Theorem 6.7 with the function f, and with the following parameters:

- $T(n) = n^k$.
- $\delta = c_0/k < 1$. (We used the fact that $k > c > c_0$.)
- $m = n^{c_0} = T^\delta$.
- $\alpha = 1/3$ and $\beta = 1/10$.

Recall that on input x, the output of G_f consists of $\bar{d}$ lists $G_f(x)_1, ..., G_f(x)_{\bar{d}}$ of m-bit strings. Consider the reconstruction algorithm R_f, instantiated with input $(1^n, n^{c_0})$ and a random seed y, and the machine $\mathcal{O}$. We claim the following.

Claim 6.8.1. *With probability at least $1 - \varepsilon$ over choice of $x \in \{0,1\}^n$,*

$$\Pr_{C\leftarrow R_f}\left[C^{\mathcal{O}}(x) \text{ prints a circuit whose truth-table is } f(x)\right] < 1 - \alpha \; . \tag{6.4}$$

Proof. Assume towards a contradiction that there exists $X \subseteq \{0,1\}^n$ of density $|X|/2^n > \varepsilon$ such that for every $x \in X$, Equation (6.4) does not hold. Then for every $x \in X$ we have that

$$\Pr_{C\leftarrow R_f}\left[\bar{C}^{\mathcal{O}}(x) = f(x)\right] \ge 1 - \alpha \; ,$$

where $\bar{C}$ is the circuit that executes the reconstructed circuit C and then evaluates its output (which is a TC^0 circuit that requires oracle access to $\mathcal{O}$) on inputs $i = 1, ..., r(n)$. It follows that

$$\mathop{\mathbb{E}}_{C \leftarrow R_f}\left[\Pr_{x \in X}\left[\bar{C}^{\mathcal{O}}(x) = f(x)\right]\right] \geq 1 - \alpha \; ,$$

and hence there is a fixed choice of seed y_n for R_f such that the deterministic circuit $C_n^\star = R_f(1^n, y_n)$ correctly computes f on at least $1 - \alpha$ of the inputs in X. In particular, the circuit family $\{C_n^\star\}$ obtained by fixing the good seed y_n as advice for R_f on each input length n satisfies $\Pr_{x \in \{0,1\}^n}\left[(C_n^\star)^{\mathcal{O}}(x) = f(x)\right] \geq 2\varepsilon/3 > \varepsilon/2$ for every $n \in \mathbb{N}$.

The size of $C_n^\star$ is at most $(n \cdot m)^{c_1} < n^{2c_0 \cdot c_1}$ and its depth is $c_1 \cdot (d/\delta^2) = (c_1/c_0^2) \cdot d \cdot k^2$, where $c_1 > 1$ is the universal constant from Theorem 6.7. It makes oracle queries of length $n + 2m < 3n^{c_0}$ to a machine running in space $c_1 \cdot \delta \cdot \log(n^k) = (c_0 \cdot c_1) \log n$. Letting $c = 2c_0 \cdot c_1$ and recalling that $C_n^\star$ can be constructed by an algorithm running in space $O(\log n)$ and using $|y_n| = O(\log n)$ bits of non-uniform advice, this contradicts our hypothesis. □

Now let us describe the deterministic algorithm that decides L. Given $x \in \{0,1\}^n$, the algorithm tries to find $i_x \in [\bar{d}]$ such that

$$\left| \Pr_{s \in [G_f(x)_i]}[M(x, s) = 1] - \Pr_r[M(x, r) = 1] \right| < \frac{1}{10} \; . \tag{6.5}$$

When it finds such an i_x, it outputs

$$\mathsf{MAJ}_{s \in [G_f(x)_{i_x}]}\{M(x, s)\};$$

if it finds no such i_x, it outputs $\perp$. To finish the proof, we need the following claim:

Claim 6.8.2. *There is a $\log(n)$-space algorithm that gets input x and satisfies the following:*

- *For at least $1 - \varepsilon$ of the inputs x, it prints i_x satisfying Eq. (6.5).*
- *Whenever it does not print i_x satisfying Eq. (6.5), it outputs $\perp$.*

Proof. We will use the "furthermore" part of Theorem 6.7. Recall that the reconstruction R_f gets input $i \in [\bar{d}]$, and for each seed $y \in \{0,1\}^{O(\log n)}$ it outputs a circuit $C_{y,i}$. As defined in the "furthermore" part, we say that R_f is successful for x with i if

$$\Pr_y\left[\text{the truth-table of } F_{x,y,i}^{\mathcal{O}} \text{ is } w_x^{(i)}\right] \geq 1 - \alpha \cdot (i/\bar{d}) \; ,$$

where $F_{x,y,i} = C_{y,i}^{\mathcal{O}}$ and $C_{y,i}$ is the output of R_f with input i and seed y.

We work in iterations $i = 2, ..., \bar{d}$. We start iteration i with the guarantee that R_f is successful for x with $i - 1$. (The assumption holds for the base case $i = 2$ since R_f is always successful with $i = 1$.) We run R_f with input i and with all possible seeds y, to verify that R_f is successful with i. For each fixed y, we check that $\mathsf{tt}\left(F_{x,y,i}^{\mathcal{O}}\right) = w_x^{(i)}$ using the reduction of printing $w_x^{(i)}$ to querying $w_x^{(i-1)}$. Whenever the algorithm printing $w_x^{(i)}$

queries $w_x^{(i-1)}$ at some $j \in [\mathrm{poly}(T)]$, we answer with $\mathsf{MAJ}_{y'}\{F^{\mathcal{O}}_{i-1,y',x}(j)\}$, relying on the hypothesis that R_f is successful for x with $i-1$. Note that we can answer oracle queries of $F_{i-1,y',x}$ to $\mathcal{O}$ by ourselves, since $\mathcal{O}$ can be computed in space $O(\log n)$.

This algorithm runs in space $O(\log T) = O(\log n)$, since at each iteration it combines constantly many algorithms that run in such space. Specifically, at each iteration i, for each seed y, the algorithm verifies that for all $j \in [\mathrm{poly}(T)]$ we have that

$$F^{\mathcal{O}}_{i,y,x}(j) = \mathsf{MAJ}_{y'}\{F^{\mathcal{O}}_{i-1,y',x}(j)\}.$$

Storing the counters for i, y, j, y' (and the number of y-s for which the verification held) can be done in space $O(\log T)$, and the string $w_x^{(i)}$ can be printed in space $O(\log T)$ while querying $w_x^{(i-1)}$. Thus, it is only left to verify that computing $F^{\mathcal{O}}_{i,y,x}(j)$ (or $F^{\mathcal{O}}_{i-1,y',x}(j)$) can be done in space $O(\log T)$. This is the case because $F_{i,y,x}$ is logspace-uniform and of size $T^{O(\delta)} \le \mathrm{poly}(T)$, so we can use the standard DFS-style emulation of circuits in bounded-space, while answering each query of $F_{y,i,x}$ to $\mathcal{O}$ by computing $\mathcal{O}$ in space $O(\delta \log T)$.[40] (And the same argument applies to $F_{x,y,i-1}$.)

By Claim 6.8.1 and the "furthermore" part of Theorem 6.7, with probability at least $1-\varepsilon$ over choice of $x \in \{0,1\}^n$, there exists $i \in \{2, ..., \bar{d}\}$ such that R_f is *not* successful for x with i (i.e., R_f is not successful for x with $i = \bar{d}$, and perhaps also with smaller values of $i < \bar{d}$). Since our iterative process only continues while R_f is successful, for the first i it encounters such that R_f is not successful with i, we have that $M(x, \cdot)$ β-distinguishes the uniform distribution over $G_f(x)_i$ from U_m.

On the other hand, if the iterative process concludes without finding a suitable i (i.e., R_f is successful for all i's), then we output $\perp$. This happens with probability at most ε over choice of input x. □

By Claim 6.8.2, with probability at least $1-\varepsilon$ over $x \in \{0,1\}^n$ the deterministic algorithm outputs $L(x)$, and whenever it does not output $L(x)$, it outputs $\perp$. ■

To relax the hypothesis from hardness on 99% of inputs to hardness on 1% of the inputs, we will use the direct-product-based hardness amplification result of Impagliazzo *et al.* [IJK+10]. For a function $f\colon \{0,1\}^n \to \{0,1\}$, let $f^{\times k}\colon \{0,1\}^{n\cdot k} \to k$ be the k-wise direct-product of f, i.e. $f^{\times k}(x_1, ..., x_k) = f(x_1) \circ f(x_2) \circ ... \circ f(x_k)$. Then:

Theorem 6.9 (approximately list-decoding the direct product code)**.** *There is a constant $c > 1$ and a probabilistic algorithm* Dec *with the following property. Let $k \in \mathbb{N}$, and $\varepsilon, \delta \in (0,1)$ be such that $\varepsilon > e^{-\delta k/c}$. Let $f\colon \{0,1\}^n \to \{0,1\}$, and let F^k such that*

$$\Pr_{(x_1,...,x_k)\leftarrow U_n^{\otimes k}}\left[F^k(x_1, ..., x_k) = f^{\times k}(x_1, ..., x_k)\right] \ge \varepsilon.$$

[40] Recall that there are $O(1)$ levels to this recursive procedure, and at each level the algorithm stores $O(\log n)$ bits. Thus, using additional $O(\log n)$ bits per level to answer the queries to $\mathcal{O}$ does not increase the space complexity above $O(\log n)$.

On input 1^n and oracle access to F^k, the algorithm Dec *prints, with probability $\Omega(\varepsilon)$, an oracle circuit F such that $\Pr_x[F^{F^k}(x) = f(x)] \geq 1 - \delta$.*

Furthermore, the algorithm Dec *is a logspace-uniform randomized oracle* $\mathbf{NC}^0$ *circuit using $O(k \log n \cdot \frac{1}{\varepsilon} \log(1/\delta))$ coins and one oracle query. The circuit F is an oracle* $\mathbf{AC}^0$ *circuit of size $\mathrm{poly}(n, k, \log(1/\delta), 1/\varepsilon)$ that uses $O(\log(1/\delta)/\varepsilon)$ non-adaptive oracle queries.*

Some of the properties stated in Theorem 6.9 (namely, the bound on the number of coins, and the fact that queries are non-adaptive) are not explicitly stated in [IJK+10]. However, these properties are immediately evident from the description of their algorithm and the resulting circuit (see [IJK+10, end of Section 1.1]). Using Theorem 6.9, we can now prove the main result for this section.

Theorem 6.10. *Assume that for every constant $c \in \mathbb{N}$ there exist constants $k, d \in \mathbb{N}$ and $\delta > 0$ and a function $f\colon \{0,1\}^n \to \{0,1\}$ that satisfies the following:*

1. ***Upper bound.*** *f is computable in logspace-uniform* $\mathbf{TC}^0$ *of depth d and size $O(n^k)$.*

2. ***Lower bound.*** *For every log-spaceadvice-uniform $(\mathbf{TC}^0)^{\mathbf{DSPACE}[c\cdot\log(n)]}$ circuit family $\{C_n\}$ of size n^c and depth $c \cdot d \cdot k^2$, and every sufficiently large $n \in \mathbb{N}$, we have*

$$\Pr_{x\in\{0,1\}^n}[C_n(x) \neq f(x)] \geq \delta\,.$$

Then, $\mathbf{BPL} \subseteq \cap_{\varepsilon>0}\mathsf{avg}_\varepsilon\mathbf{L}$.

Proof. We prove that our hypothesis implies that for every $\varepsilon > 0$, the k-wise direct-product of f (for an appropriate $k = k(\varepsilon, \delta)$) satisfies the hypothesis of Theorem 6.8.

Let $\varepsilon > 0$ be arbitrarily small, and let $c > 1$ be any constant (for which we wish to prove the hypothesis of Theorem 6.8). We instantiate our hypothesis with the constant $c' = 5c$ to obtain k, d and a function f. For $t = O(\log(1/\varepsilon)/\delta)$, let $f^{\times t}$ be the t-wise direct product of f.[41] We show that the hypothesis of Theorem 6.8 is satisfied with parameters $2k$ and d.

For the upper bound, observe that $f^{\times k}$ is computable in size $O(t \cdot n^k) = O(n^k) < n^{2k}$ and depth d. For the lower bound, assume towards a contradiction that there is a logspace-uniform $(\mathbf{TC}^0)^{\mathbf{DSPACE}[c\cdot\log(n)]}$ circuit family $\{F_n\}$ of size n^c and depth $c \cdot d \cdot (2k)^2$ that can be generated with $O(\log n)$ bits of advice such that

$$\Pr_{x\leftarrow U_n}[F_n(x) = f(x)] \geq \varepsilon/2$$

for infinitely many $n \in \mathbb{N}$. Let $\mathsf{Dec}_{\mathsf{IJKW}}$ be the algorithm from Theorem 6.9. By that theorem, there is a fixed choice of coins for $\mathsf{Dec}_{\mathsf{IJKW}}$ such that, given an answer to a single query to F_n, the algorithm $\mathsf{Dec}_{\mathsf{IJKW}}$ prints a circuit C_{n_0} satisfying

$$\Pr_{x\leftarrow U_n}[C_{n_0}^{F_n}(x) = f(x)] \geq 1 - \delta,$$

[41] On inputs whose length n is not of the form $n_0 \cdot k$ for integers n_0, k, we define $f^{\times k}$ as the evaluation of $f^{\times k}$ on the prefix of length $\lfloor n/k \rfloor \cdot k$ of the input.

where $n_0 = \lfloor n/k \rfloor$. In particular, if $\{F_n\}$ succeeds on an infinite set $S \subseteq \mathbb{N}$ of input lengths, then $\left\{C_{n_0}^{F_n}\right\}_{n\in\mathbb{N}}$ succeeds on an infinite set $\{\lfloor n/k \rfloor : n \in S\}$ of input lengths.

The only point to verify is the complexity of the circuit family $\left\{C_{n_0}^{F_n}\right\}_{n_0\in\mathbb{N}}$. Recall that the number of coins used by $\mathsf{Dec}_{\mathsf{IJKW}}$ is $O(k \log n_0 \cdot \frac{1}{\varepsilon} \log(1/\delta))$. Also note that for every n_0 such that there is $n \in S$ satisfying $n_0 = \lfloor n/k \rfloor$, we can indicate the "correct" input length n with $\lceil \log k \rceil$ bits of non-uniform advice (this is since $n = n_0 \cdot k - i$ for some $i \in \{0, ..., k-1\}$). The machine that prints $C_{n_0}^{F_n}$ gets as advice the fixed random coins for $\mathsf{Dec}_{\mathsf{IJKW}}$, the answer to the single oracle query that $\mathsf{Dec}_{\mathsf{IJKW}}$ makes to F_n, and the indication for the "correct" input length n. Along with the fact that $\mathsf{Dec}_{\mathsf{IJKW}}$ itself is logspace-uniform, we deduce that $\left\{C_{n_0}^{F_n}\right\}$ is logspace-uniform using $O(\log n)$ bits of advice.

Finally, the size of $C_{n_0}^{F_n}$ is at most $n^c \cdot n^{c_{\mathsf{IJKW}}} < (n_0)^{2c}$, relying on the fact that $n_0 = \Omega(n)$ and assuming without loss of generality that c is larger than the universal constant c_{IJKW} from Theorem 6.9. The depth of $C_{n_0}^{F_n}$ is $c \cdot d \cdot (2k)^2 + c_{\mathsf{IJKW}} < 5c \cdot d \cdot k^2$, assuming again, w.l.o.g., that c is sufficiently large. Recalling that $c' = 5c$, this contradicts our hypothesis about f. ■

Remark 6.11. In Theorem 6.10 we use the assumption (combined with hardness amplification) to obtain a function that is hard on $1-\varepsilon$ of the inputs, and deduce derandomization that succeeds on $1-\varepsilon$ of the inputs. One may wonder if assuming a function $f\colon \{0,1\}^n \to \{0,1\}^n$ that is hard on *all inputs* can yield derandomization that succeeds on all inputs (i.e., whether hardness on almost all inputs, in the sense of [CT21a], implies that $\mathsf{BPL} = \mathsf{L}$). However, such a hypothesis cannot be true in the setting of Theorem 6.10: For every candidate hard function f computable in logspace-uniform TC^0, a logspace algorithm can compute f at some fixed input (say, 1^n) and hard-wire this value into a circuit that it prints.

6.3 Unconditional Lower Bounds for Logspace-Uniform TC^0

In this section we prove Proposition 1.2. The proof closely follows the approach of Santhanam and Williams [SW13, Theorem 1.2], with only minor differences (e.g., working against uniform circuits with sub-linear advice, and some minor differences in the definition of the computational model).

We will need the following standard space-hierarchy theorem in which the machine using less space is also allowed a sub-linear amount of advice.

Claim 6.12 (space hierarchy with advice). *For every constant $c \in \mathbb{N}$ and a function $a(n) = o(n)$ there is a language in $\mathsf{DSPACE}[O(\log n)]$ that is hard for $\mathsf{DSPACE}[c \cdot \log(n)]/a(n)$.*

Proof. The machine M' for the hard language acts as follows. Associate each machine M_i with an infinite sequence of input lengths S_i. On input $x \in \{0,1\}^n$ where $n \in S_i$, the machine M' simulates $M_i(x)$ with advice x', where x' is the first $a(n)$ bits of x. Note that for every M_i and every advice sequence $\{a_n\}_{n\in\mathbb{N}}$ there are infinitely many inputs x on which $M'(x) \neq M_i(x)$. Also, M' needs space $c \cdot \log(n) + O(\log n)$, with the extra overheads used to deduce M_i from n, and to supply virtual access to the advice from the input (i.e., to store and move the heads of the machine on the input tape and the virtual advice tape). ■

We now present an alternative and more general version of Definition 1.1, referring to the "circuit-structure" function of a TC^0 circuit and allowing more bits of non-uniform advice.

Definition 6.13 (circuit-structure languages)**.** *Let $\{C_n\}$ be a TC^0 circuit family $\{C_n\}$ of size $T(n)$ and depth d. The weights function of $\{C_n\}$ is the function $f^{\mathsf{wt}}_{\{C_n\}}$ that gets input $(1^n, i, j, k)$, where $i \in [d]$ and $j, k \in [T]$, and output $w_{i,j,k}$. The thresholds function $f^{\mathsf{thr}}_{\{C_n\}}$ gets input $(1^n, i, j)$, where $i \in [d]$ and $j \in [T]$, and outputs $\theta_{i,j}$. We define a language $L^{\mathsf{wt}}_{\{C_n\}}$ such that $L_{\{C_n\}}(1^n, i, j, k, r)$ is 1 iff the r^{th} bit in the output of $f^{\mathsf{wt}}_{\{C_n\}}$ is 1. We define $L^{\mathsf{thr}}_{\{C_n\}}$ analogously.*

Recall, by Definition 6.1, that a TC^0 circuit family of size T is logspace-uniform if there are machines that compute $f^{\mathsf{wt}}_{\{C_n\}}$ and $f^{\mathsf{thr}}_{\{C_n\}}$ in space $O(\log T)$. Note that this is equivalent to deciding $L^{\mathsf{wt}}_{\{C_n\}}$ and $L^{\mathsf{thr}}_{\{C_n\}}$ in space $O(\log T)$.

Definition 6.14 (logspace-uniform circuit families with advice)**.** *We say that a circuit family $\{C_n\}$ of size $T(n)$ is* logspace-uniform with advice *$a(n)$ if there are two machines that decide $L^{\mathsf{wt}}_{\{C_n\}}$ and $L^{\mathsf{thr}}_{\{C_n\}}$ in space $O(\log T(n))$ with advice of length $a(n)$.*

The definition of log-spaceadvice-uniform from Definition 1.1 refers to the special case of Definition 6.14 when $a(n)$ is allowed to be any logarithmic function.

Note that Definition 1.1 and Definition 6.14 are not completely obvious. An alternative definition may only allow the machine to pass on the $a(n)$ bits of advice to C_n, without reading the advice or using it for its own computation; in this alternative definition, the computation that reads the advice and uses it would have only been the circuit C_n. In our particular setting (of logspace machines describing the structure of TC^0 circuits), the class defined in Definition 6.14 is stronger than the class in the alternative definition, but we can still prove unconditional lower bounds in L for this stronger class.

Theorem 6.15 (L is hard for logspace-uniform TC^0 with a bounded-logspace oracle and $n^{o(1)}$ advice)**.** *For every $k, k', d \in \mathbb{N}$, let $\mathcal{C}_{k,k',d}$ be the class of languages decidable by logspace-uniform TC^0 circuits, that on input length n can be generated with $n^{o(1)}$ bits of non-uniform advice, such that the circuits are of size n^k and depth d, and make oracle queries of length $\bar{n} = n^k$ to $\mathsf{DSPACE}[k' \cdot \log \bar{n}]$. Then, $\mathsf{L} \not\subseteq \mathcal{C}_{k,k',d}$.*

Proof. Let $L \in \mathsf{L}$ be arbitrary. Assuming that $\mathsf{L} \subseteq \mathcal{C}_{k,k',d}$, we prove that $L \in \mathsf{DSPACE}[c \cdot \log n]/n^{o(1)}$ for some fixed $c = c_{k,k',d}$, contradicting Claim 6.12. For simplicity of presentation, let us assume that all thresholds of gates in TC^0 circuits are always 0; as will be evident below, the proof of the general case is essentially identical.

Using the hypothesis, there is a family of TC^0 circuits $\{C_n\}$ of size n^k and depth d that decides L with oracle queries to $\mathsf{DSPACE}[k' \cdot \log(n^k)]$, and a machine M running in space

$C \cdot \log n$ (for some $C \in \mathbb{N}$) and advice sequence $\alpha = \{\alpha_n\}$ of length $|\alpha_n| = a(n)$ such that M with advice α decides $L^{\mathsf{wt}}_{\{C_n\}}$.

For a sufficiently small constant $\varepsilon > 0$, consider the language $L^{\mathsf{wt\text{-}unpad}}_{\{C_n\}}$, the inputs to which are of the form $(1^{n^\varepsilon}, i, j, k, r, \beta)$, and the output is the evaluation of M on (i, j, k, r) with advice β. Note that $L^{\mathsf{wt\text{-}unpad}}_{\{C_n\}} \in \mathbf{L}$.[42] Hence, using the hypothesis again, there is a family of $\mathbf{TC}^0$ circuits $\{C'_n\}$ of size $(n^\varepsilon + O(\log n) + n^{o(1)})^k < n^\delta$ and depth d such that $\{C'_n\}$ with oracle access to $\mathbf{DSPACE}[k' \cdot \log(n^{\delta \cdot k})]$ decides $L^{\mathsf{wt\text{-}unpad}}_{\{C_n\}}$, where δ may be arbitrarily small by a suitable choice of ε; we do not care about the uniformity of $\{C'_n\}$. For convenience of notation, let us reindex the family $\{C'_n\}$ such that C'_n (with α_n hard-wired in the β-variables and with oracle access to $\mathbf{DSPACE}[k' \cdot \log(n^{\delta \cdot k})]$) decides $L^{\mathsf{wt\text{-}unpad}}_{\{C_n\}}$ on inputs of the form $(1^n, i, j, k, r)$.

We now decide L in space $c \cdot \log(n)$ with $o(n)$ advice as follows. On input length n, the advice is a description of C'_n with α_n hard-wired. We rely on the following claim, which fleshes out the standard DFS-style simulation of circuits in bounded-space.

Claim 6.15.1. *We can evaluate C'_n with advice α_n and oracle access to* $\mathbf{DSPACE}[k' \cdot \log(n^{\delta \cdot k})]$ *at any given point, using space $c' \cdot (\log n)$ where $c' > 1$ is a constant that depends only on k, k', d, δ.*

Proof. We use a recursive procedure. At each node, we remember the path that led us to the node, which is a sequence of at most d choices of an edge label in $[n^\delta]$. We run the space-bounded algorithm for evaluating the gate type of the node, which is either a threshold function or a $\mathbf{DSPACE}[k' \cdot \log(n^{\delta \cdot k})]$ function, and provide it virtual access to its input gates by recursively calling the node-evaluation procedure.

Note that the recursion depth is d, and that at each level of recursion we store at most $O_{k,k',d,\delta}(\log n)$ bits (for the path and for the computation of the gate function). Thus, the overall space complexity is $O_{k,k',d,\delta}(\log n)$. $\square$

We now run the $O(k \cdot d \cdot \log n)$-space algorithm that evaluates C_n on the input x. Specifically, we use a recursive procedure for evaluation (i.e., a DFS-style simulation, similarly to the proof of Claim 6.15.1), and whenever we need to know the weights or threshold of a gate, we call the space-bounded algorithm from Claim 6.15.1. The recursion level is d, at each level we store $O_{k,k',d,\delta}(\log n)$ bits of information, and thus overall we use space $c \cdot \log n$ for some c that only depends on k and k' and d but does not depend on L. Since the advice is of length $O(n^\delta) = o(n)$, this contradicts Claim 6.12. $\blacksquare$

6.4 Scaled-Up Version: Worst-Case Derandomization

We now prove Theorem 2, which is a "scaled-up" version of Theorem 6.10. Specifically, we assume that there is a function computable in deterministic linear space that is hard for logspace-uniform $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuits of size $2^{\varepsilon \cdot n}$, and deduce derandomization of linear

[42] As usual, we can ensure that the execution stops after a logarithmic number of steps, regardless of β, using a steps counter.

space. In contrast to Theorem 6.10, in this result we only need *worst-case hardness* (rather than mild average-case hardness), the lower bound is against fully uniform models, and the conclusion is a worst-case derandomization.

Theorem 6.16. *There is a universal constant $d > 1$ such that the following holds. Assume that there are $\varepsilon > 0$ and $L \in \mathsf{DSPACE}[O(n)]$ such that L is for logspace-uniform $(\mathsf{TC}^0)^{\mathsf{ROBP}}$ circuits of depth d and size $2^{\varepsilon \cdot n}$ on all but finitely many input lengths. Then, $\mathsf{BPSPACE}[O(n)] = \mathsf{DSPACE}[O(n)]$.*

Proof. Let $L_0 \in \mathsf{BPSPACE}[O(n)]$, decided by a probabilistic machine M using space $c_0 \cdot n$. Let ε be as in our hypothesis, and let $c_{\mathsf{NW}} > 1$ be the universal constant from Theorem 7.4. Finally, let $c = (c_0 \cdot c_{\mathsf{NW}})/2\varepsilon > 1$.

Derandomization algorithm. Given $x \in \{0,1\}^n$, let $N = c \cdot n$ for a sufficiently large constant $c > 1$. We use the NW PRG from Theorem 7.4, instantiated with $\varepsilon_{\mathsf{NW}} = c_0/c \in (0,1)$ and with a hard truth-table f given by L on inputs of length N. The output length of the PRG is then $2^{\varepsilon_{\mathsf{NW}} \cdot N} = 2^{c_0 \cdot n}$ and its seed length is $\ell = O(N)$. On the given input $x \in \{0,1\}^n$, we output

$$\mathsf{MAJ}_{s \in \{0,1\}^\ell}\{M(x, \mathsf{NW}^f(s))\} ,$$

and since NW can be computed in space $O(N)$ and L can be computed in space $O(N)$, this derandomization algorithm runs in deterministic linear space $O(n)$.

Correctness. Assume that for some $x \in \{0,1\}^n$ it holds that

$$\left| \Pr_{r \in \{0,1\}^{2^{c_0 \cdot n}}}[M(x,r) = 1] - \Pr_{s \in \{0,1\}^\ell}\left[M(x, \mathsf{NW}^f(s)) = 1\right] \right| \geq \frac{1}{10} . \tag{6.6}$$

In this case, we show that L can be decided by logspace-uniform $(\mathsf{TC}^0)^{\mathsf{ROBP}}$ circuits of universal depth and size $2^{\varepsilon \cdot n}$. Specifically, we will rely on the following claim:

Claim 6.16.1. *There is an algorithm that gets input $x \in \{0,1\}^n$ such that Equation (6.6) holds, and random seed $y \in \{0,1\}^{O(N)}$, runs in space $O(N)$, and with positive probability over y it prints a $(\mathsf{TC}^0)^{\mathsf{ROBP}}$ circuit C_y of depth d (for some universal constant $d \in \mathbb{N}$) and size $2^{\varepsilon \cdot N}$, whose truth-table is f.*

Proof. We combine the distinguisher-to-predictor transformation from Theorem 4.2 and the reconstruction algorithm R_{NW} from Theorem 7.4.

Consider the ROBP defined by $D_x(r) = M(x,r)$, which is of width and length $2^{c_0 \cdot n}$. We interpret the seed y as a triplet $(i,j,b) \in \{0,1\}^{c_0 \cdot n} \times \{0,1\}^{c_0 \cdot n} \times \{0,1\}$, and let $D_{x,i,j,b}(r) = (D_x)_{i,j}(r) \oplus b$ (as defined in Definition 4.1). By Equation (6.6) and Theorem 4.2, for some (i,j,b) we have that

$$\Pr_{\substack{s \in \{0,1\}^\ell \\ r = \mathsf{NW}^f(s)}}[D_{x,i,j,b}(r_{>2^{c_0 \cdot n} - i}) = r_i] > \frac{1}{2} + \frac{1}{10} \cdot 2^{-c_0 \cdot n} .$$

Let us assume from now on that the seed $y = (i, j, b)$ satisfies the above. The reconstruction R_{NW} runs in space $O(N)$, gets oracle access to the predictor $D_{x,i,j,b}$, and prints a constant-depth oracle $\mathbf{TC}^0$ circuit C_{NW} of size $2^{c_0 \cdot c_{\mathsf{NW}} \cdot n} = 2^{(\varepsilon/2) \cdot N}$ and depth that is a universal constant $d \in \mathbb{N}$, such that $C_{\mathsf{NW}}^{D_{x,i,j,b}}$ computes f. Note that we can answer the oracle queries of R_{NW} to $D_{x,i,j,b}$ in space $O(n)$, by simulating the machine M (from configuration j, for $2^{c_0 \cdot n} - i$ steps).

Finally, recall that we want to print the $(\mathbf{TC}^0)^{\mathsf{ROBP}}$ circuit C_y, where $D_{x,i,j,b}$ is the ROBP. We can do so in space $O(N)$, since printing each of the $2^{c_0 \cdot n} - i$ layers of $D_{x,i,j,b}$ can be done in space $O(N)$ (i.e., by enumerating over all possible states, and computing the transition function of M to yield the structure of the ROBP). Since the ROBP is of width and length $2^{c_0 \cdot n}$, its total size is less than $2^{(\varepsilon/2) \cdot N}$, and hence our total output length is less than $2^{\varepsilon \cdot N}$. □

Let us explain how to use Claim 6.16.1 to contradict the hardness of L. Assume towards a contradiction that for a large enough $n \, in \mathbb{N}$, Equation (6.6) is violated for some $x \in \{0,1\}^n$, and let $N = c \cdot n$. On input 1^N, we construct a circuit for $L_n = L \cap \{0,1\}^N$ as follows:

1. Enumerate over $x \in \{0,1\}^n$.
2. Enumerate over $y = (i, j, b) \in \{0,1\}^{O(N)}$.
3. For each $z \in \{0,1\}^N$, test whether or not the circuit C_y that the algorithm from Claim 6.16.1 outputs satisfies $C_y(z) = L(z)$.
4. When finding x, y such that $C_y(z) = L(z)$ for all z, print C_y.

By our assumption, suitable x and y exist. We thus only need to verify that the algorithm above runs in space $O(N)$. Since L is computable in such space, the only non-trivial part is solving the following problem: Given input (x, y, z), compute $C_y(z)$. This can be done in space $O(N)$, by running the DFS-style simulation of the $\mathbf{TC}^0$ component of C_y, and whenever it queries the ROBP with query q, simulating M on input x, starting from state j and for i steps, where q serves as the input to the (sub) ROBP.[43] Since the circuit is of size $2^{\varepsilon \cdot N}$ and M runs in space $O(n)$, this procedure uses space at most $O(N)$. ■

6.5 More on BPL = L "On Average"

We provide evidence that it might not be harder to achieve a zero-error average-case derandomization than an average-case derandomization.

Specifically, we prove that for every distribution D over ROBPs there is a natural related distribution D' over ROBPs such that average-case derandomization with respect to D' implies a zero-error average-case derandomization with respect to D. (Jumping

[43] To be more specific, some gates in the $\mathbf{TC}^0$ circuit are not threshold gates, but are oracle gates. We simulate the ROBP that the gate computes precisely as we would simulate a threshold function that a gate computes, giving it virtual access to its inputs by space-bounded composition.

ahead, when D is uniform over a set of ROBPs, then D' is the uniform distribution over these ROBPs with a random start state.) We make the parameters precise below:

Claim 6.17 (from average-case derandomization to zero-error derandomization)**.** *For every positive integers n, any $\varepsilon, \delta > 0$, and every distribution D over ROBPs of length n and width n, there exists a distribution D' over ROBPs of length n and width n such that the following holds. If there exists a logspace algorithm $\mathcal{A}'$ such that*

$$\Pr_{B \leftarrow D'}[\mathcal{A}'(B) \in (\mathbb{E}[B] \pm \varepsilon)] \geq 1 - \delta,$$

then there is a logspace algorithm $\mathcal{A}$ such that for every $B \in \operatorname{supp}(D)$,

$$\mathcal{A}(B) \in (\mathbb{E}[B] \pm 6n \cdot \varepsilon) \cup \{\perp\},$$

and moreover, $\Pr_{B \leftarrow D}[\mathcal{A}(B) = \perp] \leq n^2 \cdot \delta$.

This claim is a simple consequence of "certified derandomization" for **prBPL**, a topic with recent interest [CH22; GRZ23; PRZ23]. We use the test of [PRZ23], as it gives a particularly simple characterization.

For a branching program B of length n and width n, for every state v let $p_{v\rightarrow}$ be the probability over a uniform input of reaching the accept state from vertex v. Then:

Lemma 6.18 ([PRZ23])**.** *Consider a set of estimates $\{\widetilde{p_{v\rightarrow}} \in (0,1)\}_v$ such that:*

1. *For every node v in the final layer n it holds that $\widetilde{p_{v\rightarrow}} = p_{v\rightarrow} \in \{0,1\}$.*
2. *For every node v in any non-final layer it holds that*

$$\left| \widetilde{p_{v\rightarrow}} - \frac{\widetilde{p_{v_0\rightarrow}} + \widetilde{p_{v_1\rightarrow}}}{2} \right| \leq \varepsilon \ ,$$

   where $v_0 = B[v,0]$ and $v_1 = B[v,1]$.

Then, it holds that $|\widetilde{p_{v_{st}\rightarrow}} - p_{v_{st}\rightarrow}| \leq 6n\varepsilon$.

From this, we can define D in terms of these tests.

Proof of Claim 6.17. We describe how D' is sampled. Given $B \leftarrow D$, let $\{T_v\}$ be subprograms of B, where T_v has start vertex v (and is otherwise identical to B, in particular with accept vertex v_{ac}). Then, let D' output a random such T_v.

Given an algorithm $\mathcal{A}'$ with the assumed guarantee over D', our algorithm $\mathcal{A}(B)$ constructs $\{T_v\}$ in logspace, lets $\widetilde{p_{v\rightarrow}} = \mathcal{A}'(T_v)$ for every v, and then verifies that these estimates satisfy the two conditions of Lemma 6.18. These conditions can be verified in logspace by re-computing values $\widetilde{p_{v\rightarrow}}$ as necessary. Finally, if all conditions are satisfied the algorithm returns $\widetilde{p_{v_{st}\rightarrow}}$, and otherwise returns $\perp$. The fact that any returned value satisfies the error bound is immediate from Lemma 6.18. Moreover, with probability at least $1 - n^2\delta$ over a uniform B, $\mathcal{A}'(T_v)$ is within ε of $\mathbb{E}[T_v]$ for every v. ∎

7 Derandomization with Minimal Memory Overhead

We first introduce and state our primary technical tool:

7.1 Technical Tool I: A Space-Efficient PRG for Adaptive ROBPs

Recall that in the standard ROBP model, the bits to be read are determined according to the *layer* of the BP. In an *adaptive* read-once model, defined also in Section 3.1, each computation path of the branching program can read the bits of input $r \in \{0,1\}^n$ in a different order, as long as each bit is read exactly once. That is, from each intermediate state, the two outgoing edges are also labeled with an index from $[n]$.

As proved in [DT23], randomized algorithms can be transformed to read the random bit at the index corresponding to their state upon reading, and this does not change their behavior with true randomness. Moreover, this transformed machine can be modeled as an AOBP.

Lemma 7.1 ([DT23])**.** *Given a randomized space-S machine M, there is a randomized oracle machine $\bar{M}$ that works as follows. $\bar{M}$ runs in space $S + O(\log S)$, and whenever $\bar{M}$ queries a random bit while in configuration τ, it queries the random oracle at position τ. Moreover, for every $x \in \{0,1\}^n$ it holds that*

$$\Pr_r[M(x,r) = 1] = \Pr_{r'}[\bar{M}^{r'}(x) = 1].$$

Finally, $\bar{M}$ on input x can be computed by an AOBP of length and width 2^S.

Proof. We define $\bar{M}$ that works the same as M but queries its randomness according to the current configuration. More formally, $\bar{M}$ is an oracle machine that on input $x \in \{0,1\}^n$ and oracle access to $r' \in \{0,1\}^S$, simulates M on x, and whenever M enters a state that flips a random coin, $\bar{M}$ queries the oracle r' in a location corresponding to the current contents of its worktape. That is, whenever a random coin is flipped, $\bar{M}$ writes its current configuration to the oracle tape, and uses it to query a bit in $r' \in \{0,1\}^S$.[44] It is a standard fact that $M_x(r)$ can be computed by an ROBP of length and width S. In D_0, the random bits are read not in the standard order, but in an order determined by the previous steps. As observed in [DT23], along each computation path, each bit of r will be read exactly once.[45] ■

Very recently, Chen, Lyu, Tal, and Wu [CLT+23] gave the first nontrivial PRG for the adaptive model, obtaining a poly-logarithmic seed length.[46]

[44] For the sake of readability, we assume that the number of configurations is exactly S, although it is slightly larger. This can be easily addressed without changing any conclusions.

[45] Again, since the number of configurations is slightly larger than S, namely $S \cdot \mathrm{polylog}(S)$, the input to the adaptive BP is slightly larger than its length. However, we can easily make it the same without any change in the theorem's statement.

[46] We note that even a seed length of n^η, where η is an arbitrarily small constant, would have sufficed for us.

Theorem 7.2 ([CLT+23]). *For any integers $n, w \geq 1$, and any $\varepsilon > 0$, there is an explicit PRG $G^{\mathsf{adp}}\colon \{0,1\}^d \to \{0,1\}^n$ that ε-fools length-n, width-w, AOBPs, where $d = O(\log n \cdot \log^2(nw/\varepsilon))$.*

The [CLT+23] construction is based on the Forbes–Kelley framework [FK18]. Next, we show that their PRG is highly space-efficient. Since we will only care about the $w = n$ case, for simplicity, we will analyze the $w = \mathrm{poly}(n)$ regime.

Claim 7.3. *When $w = \mathrm{poly}(n)$, the mapping of $(s, i) \in \{0,1\}^d \times [n]$ to $G^{\mathsf{adp}}(s)_i$ is computable in $O(\mathrm{loglog}(n/\varepsilon))$ space.*

Proof. First, we give the construction of G^{adp}, which is essentially the Forbes–Kelley generator given in [FK18] for fooling polynomial-width arbitrary-order ROBPs. For $k, r = O(\log(n/\varepsilon))$, let $D_0, \dots, D_{r-1}$ denote r independent copies of a k-wise independent distribution over $\{0,1\}^n$, and let $T_0, \dots, T_{r-1}$ denote r independent copies of a k-wise independent distribution over $\{0,1\}^n$ (also independent of the D_i-s). Let G_0 be the trivial PRG that outputs 1^n, and for $i > 0$, let

$$G_{i+1} = D_i + T_i \wedge G_i,$$

where $\wedge$ denotes bitwise AND and $+$ denotes addition over $\mathbb{F}_2^n$. We let $G^{\mathsf{adp}} = G_r$. Thus, a seed s comprises $2r$ strings, each samples from a k-wise distribution. Indeed, this takes $O(rk \log n)$ bits.

Given s, let $(d_0, \dots, d_{r-1}, t_0, \dots, t_{r-1})$ be the corresponding samples from the k-wise independent distributions. Unfolding the recursion, we have

$$G_r(s) = d_{r-1} + \sum_{\ell=0}^{r-1} d_{\ell-1} \wedge \left(\bigwedge_{z=\ell}^{r-1} t_z \right),$$

where we denote $d_{-1} = 1^n$.

By Claim 3.17, we know that given $0 \leq j \leq r-1$, s, and $\ell \in [n]$, each bit $d_j[\ell]$ (and likewise $t_j[\ell]$) can be computed in $O(\log k + \mathrm{loglog}\, n) = O(\mathrm{loglog}(n/\varepsilon))$ space. By composition of space-bounded algorithms, Proposition 3.5, each output bit of $G_r(s)$ can be computed in space

$$O\Big(\log r + \mathrm{loglog}\, \frac{n}{\varepsilon} + \log d\Big) = O\Big(\mathrm{loglog}\, \frac{n}{\varepsilon}\Big). \quad \blacksquare$$

7.2 Technical Tool II: NW With Deterministic Reconstruction

For both results, we require a version of the Nisan–Wigderson reconstructive PRG. Our presentation and proof closely follows [DT23, Theorem 5.1], except that we incorporate the uniform deterministic reconstruction of Theorem 5.1. We assume we have access to a bit-predictor, rather than a distinguisher, as we convert from the latter to the former in several different ways.

Theorem 7.4 (NW PRG with deterministic TC^0 reconstruction)**.** *There exists a universal constant $c_{\mathsf{NW}} > 1$ such that for every sufficiently small constant $\varepsilon_{\mathsf{NW}} > 0$ the following holds. There is an algorithm* NW *computing*

$$\mathsf{NW}^f \colon \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\cdot \log N} \to \{0,1\}^M$$

such that for any $f \in \{0,1\}^N$ and for $M = N^{\varepsilon_{\mathsf{NW}}}$, we have the following.

1. ***Efficiency.*** *On input s and $i \in [M]$, $\mathsf{NW}^f(s)_i$ can be computed in space $(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log N$.*
2. ***Reconstruction.*** *There is a deterministic space $O(\log N)$ algorithm R that, given oracle access to f and oracle access to a $\frac{1}{M^2}$ previous bit predictor[47] P for NW^f, prints a constant-depth oracle circuit C of size $M^{c_{\mathsf{NW}}}$ that has majority gates, makes non-adaptive queries, and satisfies the following: $C^P(x) = f_x$.*

Note that the above instantiation of the NW PRG is different than the one given in Theorem 6.6. There, the algorithm generating the decoding circuit used a short random string, and the circuit had query access to f and to an ROBP *distinguisher* rather than to a next-bit-predictor.

Proof. The generator is identical to [DT23, Theorem 5.1], except that we use the code of Theorem 5.1 (so that we can support deterministic reconstruction). Let $\rho = 1/M^2$ be the advantage of the predictor. Let $\bar{f}$ be the encoding of f by the code from Theorem 5.1 with

$$k = N, \quad \varepsilon = \rho, \quad \gamma = \varepsilon_{\mathsf{NW}}, \quad \delta = 0.1$$

and note that $\bar{f}$ is of length

$$\bar{N} = \mathrm{poly}(N/M)^{1/\varepsilon_{\mathsf{NW}}} = N^{c/\varepsilon_{\mathsf{NW}}}$$

for some universal $c > 1$. Without loss of generality, assume $\bar{N}$ is a power of two. Next, let

$$S_1, \ldots, S_k \subseteq [d]$$

be the logspace-computable $(\ell = \log \bar{N}, \alpha\ell)$ design of Theorem 3.16 with $\alpha = (c'/c)\varepsilon_{\mathsf{NW}}{}^2$, where c' is the universal constant in that theorem. With this choice we have

$$k = 2^{(\alpha/c')\ell} = M, \qquad d = (c'/\alpha)\ell = \frac{c^2}{\varepsilon_{\mathsf{NW}}{}^3} \log N.$$

Then we define the generator as follows. Given $s \in \{0,1\}^d$, for $i \in [M]$ let

$$\mathsf{NW}^f(s)_i = \bar{f}_{\mathsf{Des}(s,i)}.$$

Output Complexity. As our construction is identical to that of [DT23], the output complexity follows from their analysis, and from the fact that the code in Theorem 5.1 is encodable in space $O(\log N)$.

[47] The theorem also holds with identical parameters in the case that P is a next bit predictor.

Reconstruction. The algorithm R works as follows. Given $P : \{0,1\}^{i-1} \to \{0,1\}$ satisfying

$$\Pr_{x \leftarrow U_s}\left[P(\mathsf{NW}^f(x)_{>M-i}) = \mathsf{NW}^f(x)_{M-i}\right] \geq \frac{1}{2} + \rho$$

let $T = S_{M-i}$ and write this as

$$\Pr_{(x_T, x_{T^c}) \leftarrow U_s}\left[P(\mathsf{NW}^f(x_T \circ x_{T^c})_{>M-i}) = \bar{f}_{x_T}\right] \geq \frac{1}{2} + \rho.$$

The algorithm R then enumerates over assignments to x_{T^c} and finds a fixed $z \in \{0,1\}^{|T^c|}$ such that

$$\Pr_{x \leftarrow U_\ell}\left[P(\mathsf{NW}^f(x \circ z)_{>M-i}) = \bar{f}_x\right] \geq \frac{1}{2} + \rho,$$

which it can do in deterministic space $O(d + \log N) = O(\log N)$ by the explicitness of Theorem 5.1 and Theorem 3.16. Finally, given z we can construct in logspace a circuit for $\mathsf{NW}^f(y \circ z)_j$ for every $j > M - i$ using oracle queries to f, and moreover this circuit is an oracle $\mathbf{AC}^0$ circuit of size $2^{\alpha\ell}$. Thus, we can construct an oracle $\mathbf{AC}^0$ circuit C of size $M \cdot 2^{\alpha\ell}$ such that

$$\Pr_{x \leftarrow U_\ell}\left[C^P(x) = \bar{f}_x\right] \geq \frac{1}{2} + \rho.$$

Finally, we apply the algorithm Dec of Theorem 5.1 with $x = f$. We enumerate over random strings $y \in \{0,1\}^{O(\log N)}$ and indices $i \in \{0,1\}^{O(\log N)}$, and find the lexicographically first $C_{y,i}$ such that, letting $w = \mathsf{tt}(C^P)$, we have

$$\mathsf{tt}(C^w_{y,i}) = f.$$

Such a circuit always exists by our choice of parameters. Moreover, $C_{y,i}$ is a constant depth oracle circuit with majority gates of size $\mathrm{poly}(N^{\varepsilon_{\mathsf{NW}}} \cdot M)$, and thus the final oracle circuit that, on input j, computes $C^w_{y,i}$ and answers oracle queries to w using the circuit C^P is of size $\mathrm{poly}(N^{\varepsilon_{\mathsf{NW}}} \cdot M \cdot 2^{\alpha\ell}) = M^{c_{\mathsf{NW}}}$ where we choose c_{NW} to be a sufficiently large constant. ■

Converting Distinguishers to Previous Bit Predictors. In all cases, our correctness proof will need to transform a distinguisher for a PRG into a previous-bit predictor. We do this in three ways:

- Existentially, as such a transformation is always possible.
- If we have a PRG for the that is able to "fool the hybrid argument", as in Section 2.2.2 we can find a previous bit predictor deterministically using this PRG.
- If the distinguisher is an ROBP, we can perform this transformation in deterministic logspace via Theorem 4.2.

Lemma 7.5 ([Yao86]). *For an arbitrary distribution D over $\{0,1\}^n$ and circuit $C : \{0,1\}^n \to \{0,1\}$ of size s, if $|\mathbb{E}[C(U_n)] - \mathbb{E}[C(D)]| = \delta$, there exists a circuit P of size $s + O(1)$ that is a δ/n-previous-bit-predictor of D.*

In the case that we have a PRG that fools the distinguisher, we can find a previous bit predictor in this fashion. We let $M = N^{\varepsilon_{\mathsf{NW}}}$ and $s = (c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \log N$ be as in the statement of Theorem 7.4.

Lemma 7.6. *For every $\delta > 0$, there is a deterministic algorithm that works as follows. The algorithm is given oracle access to:*

- $f \in \{0,1\}^N$.
- *A δ-distinguisher $D : \{0,1\}^M \to \{0,1\}$ for $\mathsf{NW}^f \colon \{0,1\}^s \to \{0,1\}^M$, where NW is the generator of Theorem 7.4 with some constant $\varepsilon_{\mathsf{NW}} > 0$, so $s = O(\log N)$.*
- *A $(\delta/2M)$-HSG $H \colon \{0,1\}^t \to \{0,1\}^M$ for TC^0 circuits of size $O(2^s \cdot M)$ with oracle access to D.*

The algorithm runs in space $t + O(\log(Nt))$ outputs $z \in \{0,1\}^{M-i}$ and $b \in \{0,1\}$ such that

$$\Pr_{x \leftarrow \mathsf{NW}^f}[D(z \circ x_{>i}) \oplus b = x_i] \geq 1/2 + \delta/2M.$$

Proof. For $i \in [M]$ and $b \in \{0,1\}$, let $P_{i,b} \colon \{0,1\}^i \to \{0,1\}$ be the circuit defined as follows

$$P_{i,b}(z) = \begin{cases} 1 & \Pr_{x \leftarrow U_s}\left[D\big(z \circ \mathsf{NW}^f(x)_{>i}\big) \oplus b = \mathsf{NW}^f(x)_i\right] > \frac{1}{2} + \frac{\delta}{2M} \\ 0 & \text{o.w.} \end{cases}$$

i.e., $P_{i,b}$ accepts strings z such that fixing the first i bits of D to z causes the circuit to predict NW^f with non-negligible advantage. By Yao's unpredictability lemma and an averaging argument, there is some i, b for which $\mathbb{E}[P_{i,b}] \geq \delta/2M$. Next, we claim that P is efficient:

Claim 7.7. *For every i, b, $P_{i,b}$ can be computed by TC^0 circuits of size $O(2^s \cdot M)$ with oracle access to D.*

Proof. We construct the circuit as follows. For every $x \in \{0,1\}^s$, the circuit hardcodes $y^x = \mathsf{NW}^f(x)$, and computes

$$\sum_{x \in \{0,1\}^s} \mathbb{I}[D(z \circ y^x_{>i}) \oplus b = y^x_i]$$

and accepts if this sum is greater than $2^s \cdot (1/2 + \delta/2M)$. Then it is easy to see that $P_{i,b}$ can be implemented by TC^0 circuits of size at most $O(2^s \cdot M)$, with oracle access to D. ■

In addition, we can evaluate $P_{i,b}$ in small space:

Claim 7.8. *For every i, b, the mapping $z \to P_{i,b}(z)$ can be computed in space $O(\log N)$ with oracle access to D.*

Proof. This follows as we can compute NW^f in space $O(\log N)$ by Theorem 7.4, and we can compute the sum in space $O(s) = O(\log N)$. ■

By assumption, we have oracle access to $H\colon \{0,1\}^t \to \{0,1\}^M$, which must fool $P_{i,b}$ up to error $\delta/2M$ for every i, b. By enumerating over i, b and seeds $x \in \{0,1\}^t$, we can evaluate $P_{i,b}(H(x))$ in space $O(\log N)$. Thus, we can find i, b and $z = H(x)$ where $P_{i,b}(z) = 1$, and hence $P\colon \{0,1\}^{M-i} \to \{0,1\}$ where

$$P(x) = D(z \circ x) \oplus b$$

is a $\delta/2M$-previous-bit-predictor for NW^f, and hence output (z, b) as required. ■

7.3 Derandomization From Nonuniform Assumptions

We are now ready to prove our derandomization result that doubles the memory footprint from (standard) non-uniform hardness assumptions, as stated in Theorem 3. The construction essentially follows [DT23], wherein they constructed a suitable PRG by a composition of two "low-cost" PRGs G_1, the NW PRG, and G_2. In [DT23], G_2 was a cryptographic PRG. Here, we replace the (candidate) cryptographic PRG with the (explicit) PRG of Section 7.1.

Theorem 7.9 (derandomization that doubles the memory footprint)**.** *There exists a universal constant $c > 1$ such that for any two constants $\varepsilon \in (0,1)$ and $C \in \mathbb{N}$ the following holds. Assume that there exists*

$$L^{\mathsf{hard}} \in \mathbf{DSPACE}\left[\frac{C + 1 + \varepsilon + \delta}{2} \cdot n\right]$$

for some constant $\delta > 0$ that is hard for $\mathbf{TC}^0$ circuits of size $2^{\varepsilon \cdot n}$ with non-adaptive oracle access to algorithms that get $2^{n/2}$ bits of non-uniformity and run in space $\frac{C+1+\varepsilon}{2} \cdot n$.

Then, for $S(n) = C \cdot \log n$, we have that

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[2S + \left(\frac{c}{\varepsilon} + \delta\right) \log n\right].$$

Proof. Let $L \in \mathbf{BPSPACE}[S(n)]$, and let M be a randomized space-S machine that decides L. Let $\bar{M}$ be defined as in Lemma 7.1. Set $N = n^2$ and $\varepsilon_{\mathsf{NW}} = \frac{\varepsilon}{2c_{\mathsf{NW}}}$, and let $f \in \{0,1\}^N$ be the truth table of L^{hard} on inputs of size $\ell = \log N$. Let

$$\mathsf{NW}^f\colon \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\ell} \to \{0,1\}^d$$

be the NW PRG with $\mathbf{TC}^0$ reconstruction, given in Theorem 7.4, for d to be determined soon. Set $\bar{N}$ to be the number of seeds of NW^f, namely $\bar{N} = N^{c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}}$. Let

$$G^{\mathsf{adp}}\colon \{0,1\}^d \to \{0,1\}^{n^C}$$

be the generator from Theorem 7.2, instantiated with error $\varepsilon = 1/10$ and $w = n^C$, so $d = O(\log^3(n^C))$. Note that for a large enough n, the output of NW^f in Theorem 7.4 is much larger than d, but we can truncate accordingly.[48]

Our PRG G^f is the concatenation of the two PRGs above. That is, given $s \in [\bar{N}]$, we output

$$G^f(s) = G^{\mathsf{adp}}\big(\mathsf{NW}^f(s)\big).$$

Our final algorithm A on input x enumerates over $s \in [\bar{N}]$ and outputs

$$\mathsf{MAJ}_{s\in[\bar{N}]}\Big\{\bar{M}^{G^f(s)}(x)\Big\}.$$

Space complexity of A. The algorithm enumerates over seeds $s \in [\bar{N}]$, while also maintaining an integer counter in $[\bar{N}]$ for the majority outcome. For every fixed seed s it simulates $\bar{M}$ on the input x with oracle access to $G^f(s)$. The oracle is implemented by space-bounded composition of G^f, and of the machine for L^{hard}. The exact computation is done in [DT23, Theorem 5.5]. Denoting by S' the space complexity of computing f, the computation in [DT23] amounts to

$$\underbrace{\log \bar{N}}_{\text{enumerating } s} + \underbrace{S + O(\log S)}_{\bar{M}} + \underbrace{\frac{c_0}{\varepsilon_{\mathsf{NW}}} \cdot \log N}_{\text{computing NW}} + \underbrace{S'}_{\text{computing } f} + \underbrace{\log \bar{N}}_{\text{counting the outcomes of } M} + \underbrace{c_0 \cdot (\log N + \log(N^{\varepsilon_{\mathsf{NW}}}))}_{\text{composition overhead}} \leq \left(2C + \frac{c}{\varepsilon} + \delta\right) \cdot \log n$$

space, for some universal constants $c_0, c > 1$.

Correctness of A. Fix any $x \in \{0,1\}^n$, and let B be the AOBP of length and width n^C such that $B(r) = \bar{M}^r(x)$, as in Lemma 7.1. Recall that $\mathbb{E}_r[B(r)] = \mathbb{E}[M(x,r)]$, and moreover $|\mathbb{E}_r[B(r)] - \mathbb{E}_r[B \circ G^{\mathsf{adp}}(r)]| \leq 1/10$. Let $D = B \circ G^{\mathsf{adp}}$. Finally, assume towards a contradiction that

$$\big|\mathbb{E}[D] - \mathbb{E}\big[D \circ \mathsf{NW}^f\big]\big| > \frac{1}{10}.$$

By Lemma 7.5 and Theorem 7.4, there exists a $\mathbf{TC}^0$ circuit C of size $N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} < N^{\varepsilon}$ such that $C^D(x) = f_x$. To get a contradiction and conclude that A decides L, it is left to determine the complexity of D.

Claim 7.9.1. *D can be computed by a machine running in space $\frac{C+1+\varepsilon}{2} \log N$ with n bits of advice.*

Proof. The claim readily follows from the efficient evaluation method described in [DT23, Claim 5.6], using that G^{adp} is highly space-efficient (Claim 7.3). ☐

[48]Moreover, we can work with milder parameters since we only need a polynomial stretch, however this would not change the final derandomization result.

■

By a standard padding argument, we can conclude:

Corollary 7.10. *Under the assumption and notation of Theorem 7.9, for any $S = \Omega(\log n)$, we have that*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[\left(2 + \frac{c/\varepsilon + \delta}{C}\right) \cdot S\right].$$

In particular, for every $\tau > 0$, and $\varepsilon, \delta > 0$, there is a sufficiently large $C = C(\varepsilon, \delta, \tau)$ such that if the assumption of Theorem 7.9 holds w.r.t. C, ε, δ, then then for every $S = \Omega(\log n)$, we have that

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}[(2 + \tau)S].$$

The work of [DT23] showed that space-bounded algorithms with advice can simulate TC^0 computations as long as the circuit's size is not too large, so hardness against small TC^0 with non-adaptive oracle access to space-bounded algorithms with advice is implied by hardness against space-bounded algorithm with (a slightly longer) advice. Formally:

Claim 7.11 (Claim 5.8 of [DT23])**.** *Let L be a language that on inputs of length n can be computed by a constant-depth threshold circuit of size $2^{\varepsilon n}$ with non-adaptive oracle access to an algorithm that gets $2^{n/2}$ bits of non-uniformity and runs in deterministic space $\frac{C+1+\varepsilon}{2} \cdot n$. Then, L is also computable in deterministic space $\frac{C+1+O(\varepsilon)}{2} \cdot n$ with $2^{n/2} + 2^{\varepsilon n}$ bits of advice.*

Combining the above claim with Corollary 7.10, we obtain our first derandomization result of Theorem 3.[49] The $2 \cdot S$ term comes from simulating $\bar{M}$ (which in turn simulates M) and computing the hard function f. In [DT23], we observed that if those two computations could "share" computation space, then we can get derandomization with nearly no space overhead. One way for both computations to share space is by assuming L^{hard} is computable in mostly *catalytic* space. We can then readily get the following result, that establishes the second derandomization result of Theorem 3.

Theorem 7.12. *Assume that for a sufficiently large constant C, and some constant $\delta > 0$, there exists a language L computable in $\mathsf{CSPACE}[\delta n, (C + \delta + 1)n]$, that is hard for algorithms that run in deterministic space $C \cdot n$ with $O(2^{n/2})$ bits of advice. Then, for $S(n) = \Omega(\log n)$, we have that*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[\left(1 + \frac{(1+\delta)c}{C}\right) \cdot S\right],$$

where $c > 1$ is some fixed universal constant.

[49]To see this, suppose that Assumption 1 holds for some C, and let us show that the assumption of Theorem 7.9 holds for some C', ε, δ, and that the conclusions match. We want C', ε, δ such that the upper bound of $(C+1) \cdot n$ is smaller than the required upper bound $\frac{C'+1+\varepsilon+\delta}{2} \cdot n$ and the lower bound $C \cdot n$ is larger than the lower bound $\frac{C'+1+O(\varepsilon)}{2}$; taking $\varepsilon > 0$ to be a sufficiently small constant, and taking $\delta = 3 + O(\varepsilon)$ and $C' = 2C - O(\varepsilon) - 1$ satisfies these constraints. By Claim 7.11, the lower bound holds for constant-depth threshold circuits of size $2^{\varepsilon \cdot n}$ with non-adaptive oracle access to an algorithm that gets $2^{n/2}$ bits of non-uniformity and runs in deterministic space $\frac{C'+1+\varepsilon}{2} \cdot n$. The conclusion of Corollary 7.10 is that for $S = \Omega(\log n)$ we have $\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}[(2 + \frac{c/\varepsilon+\delta}{C'}) \cdot S]$, and the conclusion of Theorem 3 follows since $\frac{c/\varepsilon+\delta}{C'} = \frac{c/\varepsilon+\delta}{2C-O(\varepsilon)-1} < c'/C$ for a universal constant $c' > 1$.

7.4 Derandomization From Hardness of Compression

We now show that our uniform assumption, combined with an exponential stretch PRG against circuits, can be used to derive minimal-memory derandomization.

Theorem 7.13. *Suppose that Assumption 3 and Assumption 2 hold. Then, for $S(n) = C \cdot \log n$, we have that*

$$\mathsf{BPSPACE}[S] \subseteq \mathsf{DSPACE}\left[2S + \left(\frac{C}{\varepsilon} + \delta\right) \log n\right].$$

Proof. Let $L \in \mathsf{BPSPACE}[S(n)]$, and let M be a randomized space-S machine that decides L. Let $\bar{M}$ be defined as in Lemma 7.1. Set $N = n^2$, $\ell = \log N$, and $\varepsilon_{\mathsf{NW}} = \frac{\varepsilon}{2c_{\mathsf{NW}}}$. Finally, define the deterministic machine A deciding L as follows. On input x, let $f = f(x) \in \{0,1\}^N$ be the hard function on the input. Let

$$\mathsf{NW}^f \colon \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\ell} \to \{0,1\}^d$$

be the NW PRG with TC^0 reconstruction, given in Theorem 7.4, for d to be determined soon. Set $\bar{N}$ to be the number of seeds of NW^f, namely $\bar{N} = N^{c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}}$. Let

$$G^{\mathsf{adp}} \colon \{0,1\}^d \to \{0,1\}^{n^C}$$

be the generator from Theorem 7.2, instantiated with error $\varepsilon = 1/10$, so $d = O(\log^3(n^C))$. Note that for a large enough n, the output of NW^f in Theorem 7.4 is much larger than d, but we can truncate accordingly.

Our PRG G^f is the concatenation of the two PRGs above. That is, given $s \in [\bar{N}]$, we output

$$G^f(s) = G^{\mathsf{adp}}\big(\mathsf{NW}^f(s)\big),$$

and recall that it is also a function of x, unlike the generator of Section 7.3. Finally, A enumerates over $s \in [\bar{N}]$ and outputs

$$\mathsf{MAJ}_{s \in [\bar{N}]}\left\{\bar{M}^{G^f(s)}(x)\right\}.$$

Space complexity of A. As the PRG is identical to that of Theorem 3, the space complexity follows directly from that analysis.

Correctness of A. We claim that there is a space $O(C \log n)$ algorithm R such that, on every x where $A(x) \neq L_x$, outputs a compressed representation of $f(x)$ (and hence if there are infinitely many such x, we obtain a contradiction to Assumption 3). The algorithm works as follows.

Given an input $x \in \{0,1\}^n$ (where we assume that $A(x) \neq L_x$), let $B = B_x$ be the AOBP of width and length n^C such that $B(r) = \bar{M}^r(x)$, as in Lemma 7.1. Recall that

$\mathbb{E}_r[B(r)] = \mathbb{E}[M(x,r)]$, and moreover $|\mathbb{E}_r[B(r)] - \mathbb{E}_r[B \circ G^{\mathsf{adp}}(r)]| \le 1/10$. Let $D = B \circ G^{\mathsf{adp}} \colon \{0,1\}^d \to \{0,1\}$, so we obtain by assumption that

$$\left|\mathbb{E}[D] - \mathbb{E}\left[D \circ \mathsf{NW}^f\right]\right| > \frac{1}{10}.$$

Note that D can be evaluated in space $O(C \log n)$ by composition of space-bounded algorithms. We next establish a bound on its circuit complexity.

Claim 7.14. *D can be computed by an $(\mathbf{NC}^1)^{\mathsf{AOBP}}$ circuit of size $\widetilde{O}(n^C)$, and moreover the circuit makes a single* AOBP *query on every input. Consequently, D can be computed by an $\mathbf{NC}^2$ circuit of size $n' = \mathrm{poly}(n^C)$.*

Proof. Recall that the AOBP B is of size n^C. Next, we have that the mapping (s,i) to $G^{\mathsf{adp}}(s)_i$ is computable in space $O(\log\log n)$ by Claim 7.3, and thus by circuits of size $\mathrm{polylog}(n)$ and depth $O((\mathrm{loglog}\, n)^2)$. Thus, the function mapping s to $G^{\mathsf{adp}}(s)$ can be computed by an $\mathbf{NC}^1$ circuit of size $\widetilde{O}(n^C)$, and then the final top gate queries the oracle on $G^{\mathsf{adp}}(s)$. Then the final claim follows from the standard simulation of polynomial size branching program evaluation in $\mathbf{NC}^2$. ■

Finally, let $H_n \colon \{0,1\}^t \to \{0,1\}^m$ be the family of Assumption 2, where we take $m = \mathrm{poly}(n^C)$ for a sufficiently large fixed polynomial, and observe that $t = O(C \log n)$. We then apply Lemma 7.6 applied with

$$\delta = 1/10, \quad D = D, \quad f = f(x), \quad H = H_n.$$

Observe that $\mathbf{TC}^0$ circuits of size $\mathrm{poly}(n^C)$ with oracle access to D can themselves by represented as $\mathbf{NC}^2$ circuits of size $\mathrm{poly}(n^C)$, and hence H satisfies the required property.

With these parameters, the algorithm runs in space $O(C \log n)$ and outputs $z \in \{0,1\}^{d-i}$ and $b \in \{0,1\}$ such that

$$\Pr_{x \leftarrow U_s}[D(x_{<i} \circ z) \oplus b = x_i] \ge \frac{1}{2} + \frac{1}{20d}.$$

Finally, we apply the reconstruction result of Theorem 7.4. We give the algorithm R oracle access to f (which we can provide by computing $f(x)$ as needed), and next bit predictor $P(x_{<i}) = D(x_{<i} \circ z) \oplus b$. Then R runs in space $O(\log N)$ and outputs an oracle $\mathbf{AC}^0$ circuit C of size $N^{\varepsilon_{\mathsf{NW}}/2}$ such that

$$C^P(j) = f_j.$$

Our final algorithm prints the machine which, on input j, outputs $C^P(j)$. We first note that this algorithm runs in space $O(t + \log N) = O(C \log n)$ for a constant that depends on the parameters of Assumption 2.

Finally, we claim the machine printed by this algorithm has description size $O(|n|)$ and runs in space $\frac{C+1+\varepsilon}{2} \log n$. We can describe P with $O(1)$ bits for the machine M, x, and (z,d), so $n + O(1) + d = O(n)$ bits in total, and C has size $N^{\varepsilon_{\mathsf{NW}}/2} \le n$. Finally, we claim that the machine can evaluate this circuit in the desired space bound. As in Theorem 7.9, this follows from the efficient evaluation method described in [DT23, Claim 5.6]. ■

The padding argument and the simulation argument are exactly the same as in Section 7.3, so the first result of Theorem 4 readily follows. Moreover, the same holds for improving the $2 \cdot S$ factor using catalytic assumptions, so the second derandomization result of Theorem 4 follows as well.

8 BPL in CL From Certified Derandomization

Our next result is a new proof of $\mathsf{BPL} \subseteq \mathsf{CL}$ via certified derandomization.

In addition to Theorems 6.6 and 7.4, we require yet another version of the NW PRG, with the following features. First, we must be able to deterministically reconstruct a small circuit, given a *previous* bit predictor. Second, we must maintain the ability to evaluate the small circuit after erasing some bits of the original truth table. To do so, we must use the locally encodable code of Theorem 5.2. As such, our assumption is (implicitly) that our truth table is average-case, rather than worst-case hard, but this is perfectly acceptable as we will initialize our code with a catalytic tape, and in either case (our PRG is good for B or we nontrivially compress the tape) we successfully derandomize.

Theorem 8.1 (NW PRG with deterministic *approximate* reconstruction)**.** *There is a universal constant $c_{\mathsf{NW}} > 1$ such that the following holds. There is an algorithm NW computing*

$$\mathsf{NW}^f \colon \{0,1\}^{O(\log N)} \to \{0,1\}^N$$

such that for any $f \in \{0,1\}^{N^{c_{\mathsf{NW}}}}$, we have the following:

1. ***Efficiency.*** *When given $s \in \{0,1\}^{O(\log N)}$ and oracle access to f, the generator runs in space $O(\log N)$ and outputs an N-bit string $\mathsf{NW}^f(s)$.*

2. ***Deterministic Reconstruction.*** *There are deterministic space $O(\log N)$ algorithms R, T, F that act as follows.*

    - *R, given oracle access to f and oracle access to a $(1/N^2)$ previous bit predictor P for NW^f, outputs $a \in \{0,1\}^{O(\log n)}$. Moreover, there is a subset $K \subseteq \{0,1\}^{N^{c_{\mathsf{NW}}}}$ of size $N^{c_{\mathsf{NW}}}/100$ that satisfies the following.*
    - *T, given a and $i \in [N^{c_{\mathsf{NW}}}]$, determines if $i \in K$.*
    - *F, given a and oracle access to $\widetilde{f}$ such that $\widetilde{f}_K = f_K$ and oracle access to P, satisfies*

    $$\Pr_{j \in [N^{c_{\mathsf{NW}}}]}[F^{a,\widetilde{f},P}(j) = f_j] \geq 1 - N^{-c'}$$

    for a constant $c' > 0$. Moreover, F only queries $\widetilde{f}$ at locations in K.

Proof. Let c_{NW} be a sufficiently large constant to be chosen later. We use the code

$$\mathcal{C}_{\mathsf{LE}} \colon \{0,1\}^{N^{c_{\mathsf{NW}}}} \to \{0,1\}^{\bar{N}}$$

of Theorem 5.2 with $k = N^{c_{\mathsf{NW}}}$, and

$$\varepsilon = N^{-2}, \quad d = N, \quad \gamma = 1/c_{\mathsf{NW}}, \quad \delta = 0.1.$$

Let $\bar{f} = \mathcal{C}_{\mathsf{LE}}(f)$. By our choice of parameters, we have for some global constant $c > 0$ that the following hold.

- We have $\bar{N} = \mathrm{poly}(N^{c_{\mathsf{NW}}/\gamma}, N) = N^{c \cdot c_{\mathsf{NW}}^2}$ (and w.l.o.g. assume that $\bar{N}$ is a power of two).
- The encoding map has locality $L = \mathrm{poly}(N^{c_{\mathsf{NW}}\gamma}) = N^c$.

Moreover, the decoding circuits $C_{y,i}$ satisfy the following:

- Each circuit $C_{y,i}$ is specified by $|y| + |i| = O(\log N)$ bits of information.
- Each circuit $C_{y,i}$ makes $Q_{pre} = \mathrm{poly}(N, N^{c_{\mathsf{NW}}\gamma}) = N^c$ non-adaptive queries to f, independent of its input, and then on input $j \in [N^{c_{\mathsf{NW}}}]$ it makes at most $Q = \mathrm{poly}(N^{c_{\mathsf{NW}}\gamma}) = N^c$ queries to the corrupted word in order to output a single bit.
- For every corrupted codeword $w \in \{0,1\}^{\bar{N}}$ with agreement at least $1/2 + N^{-2}$ with with $\bar{f}$, there exist y, i where

$$\Pr_j\left[C_{y,i}^{f,w}(j) = f_j\right] \geq 1 - N^{-c'}$$

for some global constant $c' > 0$.

Next, let

$$S_1, \ldots, S_k \subseteq [d]$$

be the logspace-computable $(\ell = \log \bar{N}, \alpha\ell)$ design of Theorem 3.16 with $\alpha = c''/c_{\mathsf{NW}}{}^2$, where c'' is the global constant in that theorem. With this choice we have

$$k = 2^{(\alpha/c'')\ell} \geq N, \qquad d = (c''/\alpha)\ell = O(\log N), \qquad 2^{\alpha\ell} = (N^{c \cdot c_{\mathsf{NW}}{}^2})^{\alpha} = N^{c \cdot c''}.$$

Then we define the generator as follows. Given $s \in \{0,1\}^d$, for $i \in [N]$ let

$$\mathsf{NW}^f(s)_i = \bar{f}_{\mathsf{Des}(s,i)}.$$

Complexity of NW^f. The fact that NW^f can be evaluated in the claimed space follows directly from the explicitness of Theorem 5.2 and Theorem 3.16.

The Algorithm R. The algorithm R works as follows. Given $P : \{0,1\}^{i-1} \to \{0,1\}$, note that we have

$$\Pr_{x \leftarrow U_s}\left[P(\mathsf{NW}^f(x)_{>N-i}) = \mathsf{NW}^f(x)_{N-i}\right] \geq \frac{1}{2} + \frac{1}{N^2}.$$

Let $T = S_{N-i}$ and write this as

$$\Pr_{(x_T, x_{T^c}) \leftarrow U_s}\left[P(\mathsf{NW}^f(x_T \circ x_{T^c})_{>N-i}) = \bar{f}_{x_T}\right] \geq \frac{1}{2} + \frac{1}{N^2}.$$

The algorithm then enumerates over assignments to x_{T^c} and finds a fixed $z \in \{0,1\}^{|T^c|}$ such that

$$\Pr_{x_T \leftarrow U_s}\left[P(\mathsf{NW}^f(x_T \circ z)_{>N-i}) = \bar{f}_{x_T}\right] \geq \frac{1}{2} + \frac{1}{N^2}.$$

Now fixing z, note that to evaluate $P(\mathsf{NW}^f(x_T \circ z)_{>N-i})$ we must evaluate

$$h_{j,z}(x) = \mathsf{NW}^f(x_T \circ z)_j = \bar{f}_{\mathsf{Des}(x_T \circ z, j)}$$

for every j. Then let

$$E_j = \{l \in [\bar{N}] \colon \exists x, l = \mathsf{Des}(x_T \circ z, j)\}$$

be the coordinates of $\bar{f}$ that we require to evaluate h_j, and let K_j be the bits of f that are queried by the code when outputting $\bar{f}_l$ for $l \in E_j$. By the locality constraint, we have

$$|K_j| \leq L \cdot 2^{\alpha \ell} = N^{c + c \cdot c''}.$$

Thus, the total number of bits of f required to evaluate the oracle circuit C_1 such that

$$C_1^P(x) = P\big(\mathsf{NW}^f(x \circ z)_{>N-i}\big) = P(h_{N-i-1,z}(x) \circ \cdots \circ h_{N,z}(x))$$

is at most

$$N \cdot \max_j |K_j| = N^{1+c+c \cdot c''}$$

so by taking c_{NW} sufficiently large we have $N^{1+c+c\cdot c''} \leq N^{c_{\mathsf{NW}}/100}/2$. Let $w \in \{0,1\}^{\bar{N}}$ be the string where $w_x = C_1^P(x)$. Finally, we apply the decoding algorithm Dec of the code with $f = f$ and $w = w$. We enumerate over random strings $y \in \{0,1\}^{O(\log n)}$ and advice strings $i \in \{0,1\}^{O(\log n)}$ until we find $C_{y,i}$ (which always exists per the coding statement) such that

$$\Pr_j\left[C_{y,i}^{f,w}(j) = x_j\right] \geq 1 - N^{-c'}.$$

Moreover, $C_{y,i}$ makes $Q \leq N^c \leq N^{c_{\mathsf{NW}}/100}/2$ non-adaptive queries to x (that do not depend on its input j). Let these queries be K_{pre}, and let

$$K = K_{pre} \cup \left(\bigcup_j K_j\right).$$

Observe that $|K| \leq N^{c_{\mathsf{NW}}/100}$. Then R outputs

$$a = (z, y, i).$$

The Algorithm T. The algorithm T is given $a = (z, y, i)$. Observe that we can enumerate the set E_j in space $O(\log N)$ given z and j by the explicitness of Theorem 3.16, and for each $l \in E_j$, we can enumerate in space $O(\log N)$ the bits of the code used to encode the l^{th} bit of output using the explicitness of Theorem 5.2, and hence determine K_j for every j. Finally, for K_{pre}, we can determine the queries $C_{y,i}$ makes to f by running the preprocessing circuit, which does not query the corrupted codeword.

The Algorithm F. The algorithm F is given $a = (z, y, i)$ and oracle access to P and $\widetilde{f}$ such that $\widetilde{f}_K = f_K$. Then on input j, the algorithm outputs $C_{y,i}^{\widetilde{f},w}(j)$, where $w_x = C_1^P(x)$, and we construct C_1^P using z and P, and whenever we require $\bar{f}_l$ to evaluate $h_{j,z}(x)$, we compute the relevant bit of the code in space $O(\log n)$ using oracle queries to $\widetilde{f}$ and Theorem 3.16, and note that all query locations q lie in K by construction (and hence $\widetilde{f}_q = f_q$). It is clear that F only queries on locations in K, and the explicitness and space consumption follows from the explicitness of Theorem 5.2 and Theorem 3.16. ∎

We next formally define a distinguish-to-predict transformation. Note that we require the transformation to be computable without access to the distinguisher, which most (but not all, e.g. [PRZ23]) local consistency tests for ROBPs obtain.

Definition 8.2. *We say a class of circuits* $\mathbf{C}$ *has a* (black-box) distinguish to predict transformation *if there is a deterministic algorithm that, given $C \in \mathbf{C}$ of size N, outputs a collection of* $\mathbf{C}$ *circuits $P_1, \ldots, P_{\text{poly}(n)}$ of size $\text{poly}(N)$ such that for every distribution D over $\{0,1\}^n$, one of the following occurs:*

1. *D $(1/6)$-fools C.*
2. *There is i such that P_i is a $(1/6N)$-previous-bit-predictor for D.*

Observe that Theorem 4.2 is precisely the statement that there is a logspace-computable distinguish to predict transformation for ROBPs. We can then formally state the result.

Theorem 1.5 (see Section 8)**.** *Suppose a class of circuits* $\mathbf{C}$ *satisfies the following.*

1. *There is a* **CL** *algorithm that, given $C \in \mathbf{C}$ and $r \in \{0,1\}^n$, outputs $C(r)$.*
2. *There is a* **CL***-computable distinguish-to-predict transformation for* $\mathbf{C}$ *circuits.*

Then, there is a **CL** *algorithm that, given $C \in \mathbf{C}$, outputs $\mathbb{E}_r[C(r)]$ up to error $1/6$.*

Proof. Let $W = N^{c_{\mathsf{NW}}}$. We define a catalytic machine $S^{\mathbf{w}}$ that workspace $O(\log N)$ and catalytic space $W \cdot N$ (and we can see that this satisfies the requirements of the model). We divide the catalytic tape into blocks

$$\mathbf{w} = \left(\mathbf{w}^1 \circ \cdots \circ \mathbf{w}^N\right)$$

The machine iterates over $b \in [N]$ as follows. First, initialize Theorem 8.1 with $f = \mathbf{w}^b$. Next, we apply the distinguish-to-predict transformation to C and determine if for every candidate predictor P_p we have

$$\Pr_{x \leftarrow \mathsf{NW}^{\mathbf{w}^b}}[P_p(x_{>N-i}) = x_{N-i}] \leq 1/2 + 1/6N.$$

If this holds for every p, the algorithm outputs

$$\mathsf{MAJ}_s\Big\{C(\mathsf{NW}^{\mathbf{w}^b}(s))\Big\}.$$

In this case, we do not edit $\mathbf{w}$ at any point, so we clearly satisfy the requirement of catalytic computation, and by Definition 8.2, our estimate is accurate up to error $1/6$.

If for every b this does not occur, we iterate over b in increasing order and compress $\mathbf{w}^b$ as follows. For $b \in [N]$, let p be the least p such that P_p is a $(1/6N) \geq 1/N^2$ previous bit predictor for NW^{w^b}. We then call the algorithm R of Theorem 8.1 with $P = C_p$ and $f = \mathbf{w}^b$, which returns $a \in \{0,1\}^{O(\log N)}$. Let $K \subseteq [W]$ (where we implicitly consider sub-indices of $\mathbf{w}^b$) be the set determined by a, and recall that K is of density at most $W^{-.99}$ such that, for every $\tilde{\mathbf{w}}^b$ where $\tilde{\mathbf{w}}^b_K = \mathbf{w}^b_K$, we have that there exists $G \subseteq [W]$ of density at least $1 - N^{-c'}$ for some constant $c' > 0$, such that for every $j \in G$,

$$F^{h,\tilde{\mathbf{w}}^b,P}(j) = \mathbf{w}^b_j.$$

Note that $G \cup ([W] \setminus K)$ has density at least $1 - N^{-c'} - W^{-.99} \geq 1 - N^{-c'/2}$, and hence there is a subinterval

$$I_b \subseteq G \cup ([W] \setminus K)$$

of length $N^{c'/2}$. Note that:

- Editing $\mathbf{w}^b$ on the indices in I_b will not affect the behavior of F (as $I_b \subseteq [W] \setminus K$).
- F will always decode $\mathbf{w}^b$ correctly on I_b (as $I_b \subseteq G$).
- I_b can be specified by its endpoints in space $O(\log N)$.

Next, we claim that we can find such an interval.

Claim 8.2.1. *We can find such an interval in space $O(\log N)$.*

Proof. This follows from the fact that given $j \in [W]$, we can test if $j \in K$ by the algorithm T of Theorem 8.1, and can test if $j \in G$ by computing $F^{a,\mathbf{w}^b,P_p}(j)$ and comparing to $\mathbf{w}^b_j$. □

Once we have found such an interval, set

$$(\mathbf{w}^b)_{I_b} = (p \circ a \circ I_{b-1} \circ 0^*)$$

where

1. p is the index of the previous-bit-predictor P_p,
2. a is the string produced by R,
3. I_{b-1} is a pointer to the compressed interval in $\mathbf{w}^{b-1}$.

Once we have performed this operation for every $b \in [N]$, we can use the $N \cdot (N^{c''} - O(\log N)) \geq 2N$ total free bits specified by the union of the free space on each interval to compute the exact expectation of C. After we have done this, we can iterate backwards over $b \in [N]$. We load p and a and a pointer to I_{b-1} into workspace, and then for every $j \in I_b$, we let

$$\tilde{\mathbf{w}}_j^b \leftarrow F^{a,\tilde{\mathbf{w}}^b,P_p}(j)$$

and after this we have that $\tilde{\mathbf{w}}^b = \mathbf{w}^b$, and hence after finishing this loop we correctly reset the tape. The correctness and space-efficiency follow from the performance of F. ■

Corollary 8.3. $\mathbf{BPSPACE}[S(n)] \subseteq \mathbf{CSPACE}[O(S(n)), 2^{O(S(n))}]$.

Proof. Let $L \in \mathbf{BPSPACE}[S(n)]$, and let M be a randomized space-S machine that decides L, and let $N = 2^S$. The catalytic machine, on input x, construct the branching program $B(r) = M(x, r)$. We then apply Theorem 1.5 and return the corresponding answer. We can apply this theorem as ROBPs are logspace-evaluable, and we have a logspace-computable distinguish to predict transformation for ROBPs by Theorem 4.2. Correctness follows from the correctness of the randomized algorithm and Theorem 1.5. ■

References

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

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. "Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs". In: *IEEE Transactions on Information Theory* 38.2 (1992), pp. 509–516.

[AKM+20] AmirMahdi Ahmadinejad, Jonathan A. Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil Vadhan. "High-precision Estimation of Random Walks in Small Space". In: *Proc. 61 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 1295–1306.

[BCG20] Mark Braverman, Gil Cohen, and Sumegha Garg. "Pseudorandom Pseudo-distributions with Near-Optimal Error for Read-Once Branching Programs". In: *SIAM Journal on Computing* 49.5 (2020).

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. "Computing with a full memory: catalytic space". In: *Proc. 46 Annual ACM Symposium on Theory of Computing (STOC)*. 2014, pp. 857–866.

[BGG93] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. "Randomness in Interactive Proofs". In: *Computational Complexity* 3 (1993), pp. 319–354.

[BKL+16] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. "Catalytic Space: Non-determinism and Hierarchy". In: *Proc. 33 Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 47. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 24:1–24:13.

[BT06] Andrej Bogdanov and Luca Trevisan. "Average-case complexity". In: *Foundations and Trends® in Theoretical Computer Science* 2.1 (2006), pp. 1–106.

[CDS+23] Gil Cohen, Dean Doron, Ori Sberlo, and Amnon Ta-Shma. "Approximating Iterated Multiplication of Stochastic Matrices in Small Space". In: *Proc. 55 Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 35–45.

[CH22] Kuan Cheng and William M. Hoza. "Hitting Sets Give Two-Sided Derandomization of Small Space". In: *Theory Comput.* 18 (2022), pp. 1–32.

[CHH+19] Eshan Chattopadhyay, Pooya Hatami, Kaave Hosseini, and Shachar Lovett. "Pseudorandom generators from polarizing random walks". In: *Theory of Computing* 15 (2019), Paper No. 10, 26.

[CHL+23] Lijie Chen, William M. Hoza, Xin Lyu, Avishay Tal, and Hongxun Wu. "Weighted Pseudorandom Generators via Inverse Analysis of Random Walks and Shortcutting". In: *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. To appear. 2023.

[CL20] Eshan Chattopadhyay and Jyun-Jie Liao. "Optimal Error Pseudodistributions for Read-Once Branching Programs". In: *Proc. 35th Annual IEEE Conference on Computational Complexity (CCC)*. 2020, 25:1–25:27.

[CL23] Eshan Chattopadhyay and Jyun-Jie Liao. "Recursive Error Reduction for Regular Branching Programs". In: *Electronic Colloquium on Computational Complexity: ECCC* (2023).

[CLO+23] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. "Polynomial-Time Pseudodeterministic Construction of Primes". In: *arXiv preprint arXiv:2305.15140* (2023).

[CLT+23] Lijie Chen, Xin Lyu, Avishay Tal, and Hongxun Wu. "New PRGs for Unbounded-Width/Adaptive-Order Read-Once Branching Programs". In: *Proc. 50 International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 261. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 39:1–39:20.

[CRT+20] Lijie Chen, Ron D. Rothblum, Roei Tell, and Eylon Yogev. "On Exponential-Time Hypotheses, Derandomization, and Circuit Lower Bounds". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 13–23.

[CRT22] Lijie Chen, Ron D. Rothblum, and Roei Tell. "Unstructured Hardness to Average-Case Randomness". In: *Proc. 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 429–437.

[CT21a] 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)*. 2021, pp. 125–136.

[CT21b] Lijie Chen and Roei Tell. "Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost". In: *Proc. 53st Annual ACM Symposium on Theory of Computing (STOC)*. 2021, pp. 283–291.

[CT23a] Lijie Chen and Roei Tell. "Guest column: New ways of studying the $\mathsf{BPL} = \mathsf{P}$ conjecture". In: *ACM SIGACT News* 54.2 (2023), pp. 44–69.

[CT23b] Lijie Chen and Roei Tell. "When Arthur has Neither Random Coins nor Time to Spare: Superfast Derandomization of Proof Systems". In: *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 60–69.

[CTW23] Lijie Chen, Roei Tell, and Ryan Williams. "Derandomization vs Refutation: A Unified Framework for Characterizing Derandomization". In: *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. To appear. 2023.

[DMO+22] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. "Nearly Optimal Pseudorandomness From Hardness". In: *Journal of the ACM* 69.6 (2022), pp. 1–55.

[DT23] Dean Doron and Roei Tell. "Derandomization with Minimal Memory Footprint". In: *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[FK18] Michael A. Forbes and Zander Kelley. "Pseudorandom generators for read-once branching programs, in any order". In: *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2018, pp. 946–955.

[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)*. 2007, pp. 440–449.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for muggles". In: *Journal of the ACM* 62.4 (2015), 27:1–27:64.

[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)*. 1989, pp. 25–32.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. New York, NY, USA: Cambridge University Press, 2008.

[Gol11a] Oded Goldreich. "A Brief Introduction to Property Testing". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. 2011, pp. 465–469.

[Gol11b] Oded Goldreich. "Candidate one-way functions based on expander graphs". In: *Studies in complexity and cryptography*. Vol. 6650. Lecture Notes in Computer Science. Springer, Heidelberg, 2011, pp. 76–87.

[Gol11c] Oded Goldreich. "In a World of $\mathsf{P} = \mathsf{BPP}$". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*. 2011, pp. 191–232.

[Gol11d] Oded Goldreich. "Two Comments on Targeted Canonical Derandomizers". In: *Electronic Colloquium on Computational Complexity: ECCC* (2011).

[Gol18] Oded Goldreich. "On doubly-efficient interactive proof systems". In: *Foundations and Trends® in Theoretical Computer Science* 13.3 (2018).

[GRZ23] Uma Girish, Ran Raz, and Wei Zhan. "Is Untrusted Randomness Helpful?" In: *Proc. 14 Conference on Innovations in Theoretical Computer Science (ITCS)*. Ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 56:1–56:18.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. "Uniform constant-depth threshold circuits for division and iterated multiplication". In: *Journal of Computer and System Sciences* 65.4 (2002), pp. 695–716.

[Hea08] Alexander D. Healy. "Randomness-efficient sampling within NC^1". In: *Computational Complexity* 17.1 (2008), pp. 3–37.

[HH23] Pooya Hatami and William M. Hoza. "Theory of Unconditional Pseudorandom Generators". In: *Electronic Colloquium on Computational Complexity: ECCC* (2023).

[Hoz21] William M. Hoza. "Better Pseudodistributions and Derandomization for Space-Bounded Computation". In: *Proceedings of the 25th International Conference on Randomization and Computation (RANDOM)*. 2021, 28:1–28:23.

[HV06] Alexander Healy and Emanuele Viola. "Constant-Depth Circuits for Arithmetic in Finite Fields of Characteristic Two". In: *Proc. 23 Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 3884. Lecture Notes in Computer Science. Springer, 2006, pp. 672–683.

[HZ20] William M. Hoza and David Zuckerman. "Simple Optimal Hitting Sets for Small-Success RL". In: *SIAM Journal on Computing* 49.4 (2020), pp. 811–820.

[IJK+10] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. "Uniform direct product theorems: simplified, optimized, and derandomized". In: *SIAM Journal on Computing* 39.4 (2010), pp. 1637–1665.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. "Pseudorandomness for network algorithms". In: *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*. 1994, pp. 356–364.

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

[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)*. 1998, pp. 734–743.

[KM02] Adam R. Klivans and Dieter van Melkebeek. "Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses". In: *SIAM Journal on Computing* 31.5 (2002), pp. 1501–1526.

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

[LP22b] Yanyi Liu and Rafael Pass. "Leakage-Resilient Hardness v.s. Randomness". In: *Electronic Colloquium on Computational Complexity: ECCC* TR22-113 (2022).

[Mer23] Ian Mertz. "Reusing Space: Techniques and Open Problems". In: *Bulletin of EATCS* 141.3 (2023).

[Nis92] Noam Nisan. "Pseudorandom generators for space-bounded computation". In: *Combinatorica* 12.4 (1992), pp. 449–461.

[Nis93] Noam Nisan. "On Read-Once vs. Multiple Access to Randomness in Logspace". In: *Theoretical Computer Science* 107.1 (1993), pp. 135–144.

[Nis94] Noam Nisan. "$\mathsf{RL} \subseteq \mathsf{SC}$". In: *Computational Complexity* 4 (1994), pp. 1–11.

[NN93] Joseph Naor and Moni Naor. "Small-bias probability spaces: efficient constructions and applications". In: *SIAM Journal on Computing* 22.4 (1993), pp. 838–856.

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs. randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167.

[NZ96] Noam Nisan and David Zuckerman. "Randomness is Linear in Space". In: *Journal of Computer and System Sciences* 52.1 (1996), pp. 43–52.

[PP23] Aaron (Louie) Putterman and Edward Pyne. "Near-Optimal Derandomization of Medium-Width Branching Programs". In: *Proc. 55 Annual ACM Symposium on Theory of Computing (STOC)*. 2023, pp. 23–34.

[PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. "Certified Hardness vs. Randomness for Log-Space". In: *Proc. 64 Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. To appear. 2023.

[Rei08] Omer Reingold. “Undirected connectivity in log-space”. In: *Journal of the ACM* 55.4 (2008), 17:1–17:24.

[RT92] John H. Reif and Stephen R. Tate. “On threshold circuits and polynomial computation”. In: *SIAM Journal on Computing* 21.5 (1992), pp. 896–908.

[RVW02] Omer Reingold, Salil Vadhan, and Avi Wigderson. “Entropy waves, the zig-zag graph product, and new constant-degree expanders”. In: *Annals of Mathematics* 155 (2002), pp. 157–187.

[SM23] Nicollas Sdroievski and Dieter van Melkebeek. “Instance-Wise Hardness versus Randomness Tradeoffs for Arthur-Merlin Protocols”. In: *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. “Pseudorandom generators without the XOR lemma”. In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 236–266.

[SU05] Ronen Shaltiel and Christopher Umans. “Simple extractors for all min-entropies and a new pseudorandom generator”. In: *Journal of the ACM* 52.2 (2005), pp. 172–216.

[SV22] Ronen Shaltiel and Emanuele Viola. “On Hardness Assumptions Needed for “Extreme High-End” PRGs and Fast Derandomization”. In: *Proc. 13 Conference on Innovations in Theoretical Computer Science (ITCS)*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 116:1–116:17.

[SW13] Rahul Santhanam and R. Ryan Williams. “On medium-uniformity and circuit lower bounds”. In: *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*. IEEE, 2013, pp. 15–23.

[SZ99] Michael E. Saks and Shiyu Zhou. “$\mathsf{BP_HSPACE}[S] \subseteq \mathsf{DSPACE}[S^{3/2}]$”. In: *Journal of Computer and System Sciences* 58.2 (1999), pp. 376–403.

[TV07] Luca Trevisan and Salil Vadhan. “Pseudorandomness and Average-Case Complexity Via Uniform Reductions”. In: *Computational Complexity* 16.4 (2007), pp. 331–364.

[Uma03] Christopher Umans. “Pseudo-random generators for all hardnesses”. In: *Journal of Computer and System Sciences* 67.2 (2003), pp. 419–440.

[Yao86] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *Proc. 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1986, pp. 162–167.

[Zuc97] David Zuckerman. “Randomness-optimal oblivious sampling”. In: *Random Structures & Algorithms* 11.4 (1997), pp. 345–367.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il