Electronic Colloquium on Computational Complexity, Report No. 36 (2023)

# Derandomization with Minimal Memory Footprint

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

Roei Tell*
IAS and DIMACS
roeitell@gmail.com

**Abstract**

Existing proofs that deduce $\mathbf{BPL} = \mathbf{L}$ from circuit lower bounds convert randomized algorithms into deterministic algorithms with large constant overhead in space. We study space-bounded derandomization with minimal footprint, and ask what is the minimal possible space overhead for derandomization. We show that $\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[c \cdot S]$ for $c \approx 2$, assuming space-efficient cryptographic PRGs, and, either: (1) lower bounds against bounded-space algorithms with advice, or: (2) lower bounds against certain *uniform* compression algorithms. Under additional assumptions regarding the power of catalytic computation, in a new setting of parameters that was not studied before, we are even able to get $c \approx 1$.

Our results are constructive: Given a candidate hard function (and a candidate cryptographic PRG) we show how to transform the randomized algorithm into an efficient deterministic one. This follows from new PRGs and targeted PRGs for space-bounded algorithms, which we combine with novel space-efficient evaluation methods. A central ingredient in all our constructions is hardness amplification reductions in logspace-uniform $\mathbf{TC}^0$, that were not known before.

*Part of this work was done while visiting the Simons Institute for the Theory of Computing. Supported in part by the National Science Foundation under grant number CCF-1900460.

ISSN 1433-8092

# Contents

# 1 Introduction

One of the greatest challenges in complexity theory is the derandomization of efficient algorithms, or more broadly, understanding to what extent is randomness necessary or useful for algorithms. In the time-bounded setting, can we simulate any randomized algorithm by a deterministic one with a roughly similar runtime? In the space-bounded setting, can we derandomize with only a small factor blowup in space?

Classical hardness-to-pseudorandomness results tell us that under plausible circuit lower bounds, any randomized algorithm that runs in time $T$ can be be simulated deterministically by an algorithm running in time $T^c$ [NW94; IW97], and any randomized algorithm that uses $S$ space can be simulated deterministically in space $c \cdot S$ [KM02], where $c$ is a large constant. In the terminology of complexity classes, $\mathbf{BPP} = \mathbf{P}$ and $\mathbf{BPL} = \mathbf{L}$ follow from circuit lower bounds.[1]

The constant $c$ in the foregoing classical results can indeed be large to the point of impracticality, for reasons that are inherent to the proof techniques. Therefore, a natural question is whether these results can be made *more efficient*, by providing an explicit small bound on the time or space overhead in derandomization. In other words, we ask what is the *actual, fine-grained value of randomness* in various computational settings. Note that this question is likely to be relevant even when the great goal of establishing $\mathbf{BPP} = \mathbf{P}$ and $\mathbf{BPL} = \mathbf{L}$ without relying on hardness assumptions is achieved.

In recent years, starting with the work of Doron, Moshkovitz, Oh, and Zuckerman [DMO+20] and continuing with the works of Chen and Tell [CT21b; CT21a; CT22], a study of *fine-grained derandomization* led to a series of results:[2]

- $\mathbf{BPTIME}[T] \subseteq \mathbf{DTIME}[T^{2+\alpha}]$ assuming there exists a language in $\mathbf{DTIME}[2^{(1+\alpha)n}]$ that is hard for certain randomized, non-deterministic circuits of size $2^{(1-\alpha)n}$ [DMO+20]. Chen and Tell [CT21b] showed that one can get rid of the circuits' randomness by assuming that the language is batch-computable.

- $\mathbf{BPTIME}[T] \subseteq \mathbf{DTIME}[n^{1+\alpha} \cdot T]$, where $n$ denotes the length of the input, assuming that one-way functions exist, and there exists a language in $\mathbf{DTIME}[2^{k \cdot n}]$ that is hard for $\mathbf{DTIME}[2^{(k-\alpha)\cdot n}]/2^{(1-\alpha)\cdot n}$ [CT21b]. In a followup work, Chen and Tell [CT22] showed that one can forgo the cryptographic assumption and replace it with [DMO+20]-style ones.

- $\mathbf{BPTIME}[T] \subseteq \mathbf{heurDTIME}[n^{\alpha} \cdot T]$, meaning that the derandomization fails with negligible probability with respect to all efficiently-samplable distributions [CT21b]. This result follows from uniform cryptographic assumptions and certain uniform hardness assumptions for multi-bit output functions.

[1] We also have equality between the promise classes. In fact, all our results in this paper will hold for the corresponding promise classes as well, but for readability we will omit the promise problems notation.

[2] In what follows, $\alpha > 0$ is an arbitrarily small constant, but different appearances of $\alpha$ may (or should) not be the same. We refer the reader to the relevant papers for the precise statements.

- Derandomization of *interactive proof systems* with constantly many rounds, that either has a (bounded) polynomial time overhead that depends on the number of rounds, or has only $n^\alpha$ time overhead and yields a deterministic (**NP**-style) argument system [CT22].

These results are often complemented with nearly matching conditional lower bounds (i.e., lower bounds assuming certain complexity-theoretic assumptions). In addition to derandomization in nearly no cost, those results gave rise to new notions, tools, and techniques in derandomization.

**The space-bounded setting.** In this work, we study efficient *space*-bounded derandomization under hardness assumptions, asking what is the minimal possible *space* overhead for derandomization. That is, can we transform randomized algorithms into deterministic ones that use roughly the same amount of memory?

We note that unlike the time-bounded setting, wherein unconditional derandomization results would lead to lower bounds that currently seem out of reach (see, e.g., [IKW02; KI04; KMS12; Wil11; MW18; Tel19; Che19; CLW20]), in the space-bounded setting we do have unconditional partial derandomization results. Savitch's theorem [Sav70] can be extended to show that $\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[O(S^2)]$ (see also [BCP83]). Nisan [Nis92; Nis94] devised a time-efficient derandomization with a quadratic overhead in space, namely, $\mathbf{BPL} \subseteq \mathbf{DTISP}[\mathrm{poly}(n), O(\log^2 n)]$. Focusing solely on space, Saks and Zhou [SZ99] cleverly built on Nisan's work to deterministically simulate space-$S$ randomized algorithms in $\mathbf{DSPACE}[O(S^{2/3})]$. The state-of-the-art is a recent improvement by Hoza [Hoz21], giving a deterministic simulation in space $O(S^{2/3}/\sqrt{\log S})$.

Still, even *when* $\mathbf{BPL} = \mathbf{L}$ is proven, it is very likely that the minimal-footprint derandomization question would remain: What is the minimal $c$ for which

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[c \cdot S]?$$

We will give assumptions under which $c$ approaches $2$, and further assumptions under which $c$ approaches $1$! Moreover, the results in this paper are *constructive*. Namely, given a candidate hard function (and a candidate cryptographic PRG), we show how to transform the randomized algorithm into an efficient deterministic one.

We proceed to give an overview of our results.

## 1.1 Setting the stage: A tighter hypothesis and improved local list decoding

We first revisit the Klivans–van-Malkebeek result [KM02] that establishes $\mathbf{BPL} = \mathbf{L}$ from standard, nonuniform hardness assumptions. The [KM02] result, which goes along the line of [NW94], states that given a language in $\mathbf{DSPACE}[O(n)]$ that is hard for circuits

of size $2^{\varepsilon n}$, then $\mathbf{BPL} = \mathbf{L}$.[3] Can we do better? In particular, can we work with a more restricted class of circuits? We show:

**Theorem 1** (see also Theorem 5.2)**.** *Assume there exists a language $L \in \mathbf{DSPACE}[O(n)]$ that is hard for $\mathbf{TC}^0$ circuits of size $2^{\varepsilon n}$, for some $\varepsilon \in (0,1)$, with oracle access to read-once branching programs of length and width $2^{\varepsilon n}$. Then, for $S = \Omega(\log n)$,*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[O(S)].$$

While Theorem 1 is not needed for our minimal-footprint results, the main ingredient that goes into the proof of Theorem 1 is a basic component in all of our results: We give a new *hardness amplification* result in $\mathbf{TC}^0$, or equivalently, a new locally list decodable code with $\mathbf{TC}^0$ decoding. We elaborate on it in Section 2.1.

## 1.2 Black-box derandomization with minimal footprint

Our first derandomization result follows from *worst-case nonuniform hardness assumptions* and *cryptographic assumptions*. We begin with our standard hardness assumption, against small-space algorithms that use non-uniformity.

**Assumption 1** (nonuniform hardness assumption)**.** *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 $2^{n/2}$ bits of advice.*

We note that the gap of $C+1$ vs. $C$ can be replaced by any constant gap (i.e., $C+k$ vs. $C$ for any small constant $k$), at the cost of allowing a relatively minor additional overhead in the derandomization; for the precise statement, see Theorem 5.5. We note that a small gap between the space complexity of $L$ and the space for which it is hard for, is inherent for "super efficient" derandomization results merely due to space hierarchy theorems.

We continue with our cryptographic PRG.

**Assumption 2.** *There exists a polynomial-stretch PRG fooling circuits of arbitrary polynomial size, computable in logarithmic space.*[4]

One appealing candidate for a cryptographic PRG satisfying Assumption 2 is Goldreich's expander-based PRG [Gol11b], instantiated with expanders whose neighbor function is logspace-computable; we elaborate on this below. However, in the assumption we can use any cryptographic PRG with polynomial stretch, as long as its space complexity is as described.

Equipped with those two assumptions, we can state our efficient derandomization from worst case hardness assumptions.

[3] In [KM02] it is also stated that one can obtain $\mathbf{BPL} = \mathbf{L}$ from a size-$2^{\varepsilon n}$ lower bound on branching programs. The proof of this statement is not spelled out there in full detail, and as far as we understand, the branching programs referred to in the statement are non oblivious and non read once – a model that lies between $\mathbf{NC}^1$ and $\mathbf{AC}^1$. Theorem 1 gives a stronger statement.

[4] That is, for any constants $\eta$ and $C$ we have a PRG $C^{\mathsf{cry}}\colon \{0,1\}^{n^\eta} \to \{0,1\}^{n^C}$ computable in space $O(\log(n^\eta) + \log\log(n^C))$.

**Theorem 2** (see also Theorem 5.5)**.** *Suppose that Assumption 1 and Assumption 2 hold. Then, for $S(n) = \Omega(\log n)$, we have that*

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

*where $c > 1$ is some fixed universal constant.*

As the section's name suggests, the above result uses a space-efficient pseudorandom generator (along the lines of [CT21b]), which we combine with a new method to space-efficiently evaluate a space-bounded machine over the PRG's image, utilizing the machine's own configuration. See Section 2.2 for a discussion about the techniques.

**On the hardness assumptions.** The combination of two assumptions – one asserting hardness for non-uniform machines, and an additional one that is either cryptographic or follows [DMO+20] – is in line with previous works in the area (see [CT21b; CT22]). However, previous works focused on time bounded algorithms, whereas the space bounded model turns out to be more subtle, posing several additional challenges. Thus, the particular hardness assumptions that we use above are more specialized. Let us elaborate.

First, note that the hypothesized lower bound in Assumption 1 holds for probabilistic space-bounded machines. One could have hoped for a hardness assumption that is even closer to Theorem 1, namely, for *read-once branching programs* (or for $\mathbf{TC}^0$ circuits with oracle access to such programs). In the technical section we show that Assumption 1 can indeed be relaxed to a seemingly weaker, branching programs based assumption, which is a bit more involved to state (see Section 5.3.2 for details).

Secondly, our cryptographic PRG is not just an arbitrary one, but has to be logspace-computable. However, as mentioned above, we propose Goldreich's PRG [Gol11b] as a natural and well-studied candidate, that works as follows. Let $\Gamma\colon [n^C] \times [d] \to [n^\eta]$ be the neighbor function of a suitable *lossless expander*, and let $P\colon \{0,1\}^d \to \{0,1\}$ be a predicate. Then, given $s \in \{0,1\}^{n^\eta}$ and $i \in [n^C]$, we define

$$G^{\mathsf{exp}}(s)_i = P\left(s{\restriction_{\Gamma(i)}}\right),$$

where $s{\restriction_{\Gamma(i)}}$ is the restriction of $s$ to the set of right-neighbors of $i$ the lossless expander. For $\Gamma$, we use an explicit, space-efficient expander [GUV09; KT22]. We further discuss Goldreich's PRG and its security, including possible choices for $P$, in Section 5.3.1.

## 1.3 Non black-box derandomization with minimal footprint

Next, we turn to minimal-footprint derandomization under *uniform* hardness assumptions. Roughly speaking, we assume the existence of a function computable in space $(C+1) \cdot n$ that cannot be probabilistically "compressed" (even in slightly larger space) into a small Turing machine that uses only $C \cdot n$ space and computes the function. Formally:

**Definition 1.1.** *We say that $P \in \{0,1\}^\star$ is an $S$-space* compressed version of $f \in \{0,1\}^\star$ *if $P$ is a description, of length $\sqrt{|f|}$, of a Turing machine $M$ that satisfies the following: On input $x \in [|f|]$, the machine $M$ runs in space $S(|x|)$ and outputs $f_x$.*

**Assumption 3.** *For a sufficiently large constant $C$, there exists a function $f\colon \{0,1\}^\star \to \{0,1\}^\star$ mapping $n$ bits to $n^2$ bits, that is computable in space $(C+1)\cdot \log n$, and satisfies the following. For every probabilistic algorithm $R$ running in space $C\cdot\log n + O(\log n)$[5] there are at most finitely many $x \in \{0,1\}^\star$ for which*

$$\Pr\left[R(x) \text{ prints a } (C\cdot\log n)\text{-space compressed version of } f(x)\right] \geq \frac{2}{3}.$$

Again, similarly to our comments after Assumption 1, the precise difference of $C+1$ vs. $C$ is not crucial (i.e., we can use $C+k$ vs. $C$ for a fixed small universal $k$), and moreover the precise "amount of compression" can also be relaxed (e.g., compressing to $|f|^{0.01}$ instead of to $\sqrt{|f|}$); see Section 6 for the precise details.

**Theorem 3** (see also Theorem 6.5)**.** *Suppose that Assumption 3 and Assumption 2 hold. Then, for $S(n) = \Omega(\log n)$ we have that*

$$\mathbf{BPSPACE}[S] \in \mathbf{DSPACE}\left[\left(2+\frac{c}{C}\right)S\right],$$

*where $c > 1$ is some fixed universal constant.*

The derandomization algorithm in Theorem 3 does not rely on a pseudorandom generator, but instead works in a "non black-box" way that depends on the *input*. This follows an approach in a recent line of works initiated in [CT21a] (with origins dating back to [GW02; Gol11c]). As in previous works, the underlying hardness-to-randomness tradeoff is *instance-wise*, in the sense that for every space-$S$ machine $M$ and any *fixed input* $x$, if a certain machine $R_M(x)$ fails to print a compressed version of $f(x)$, then the deterministic simulation of $M$ succeeds at the particular input $x$.

**On the hardness assumption.** The hardness assumption in Theorem 3 is different than the one in Theorem 2, but the conclusion is identical. This lends additional support for the possibility of derandomization with small footprint. Moreover, assuming hardness only for uniform algorithms (as in Theorem 3) is preferable, and the notion of hardness on all but finitely many inputs is necessary for derandomization and was used in several recent works (see, e.g., [CT21a; LP22a; LP22b]).

Nevertheless, the particular notion of hardness of *compressing* $f(x)$ is non standard, and we elaborate on it. Recall that, by Kolmogorov-complexity-type arguments, almost all strings do not have *any* concise representation, let alone one that represents a space-bounded machine. (In particular, since such a representation does not exist, then certainly it is impossible to efficiently find it as in Assumption 3.) The crux of the assumption is

[5] The constant hidden in the $O()$ does not depend on $C$.

that such representations are infeasible to find even for the outputs of the efficiently-computable function $f(x)$. We also note that an assumption reminiscent of "hardness of compressing $f(x)$ on all but finitely many inputs $x$" was recently used to *characterize time-bounded derandomization* (i.e., it is equivalent to the statement $\mathbf{prBPP} = \mathbf{prP}$); see [LP22a] for precise details.

Lastly, since the underlying hardness-vs.-randomness tradeoff is instance-wise, the statement of Theorem 3 is robust, in the following sense: If the hardness holds not on all $n$-bit inputs, but rather only on $1-\mu(n)$ fraction of the $n$-bit inputs over some distribution $\mathbf{x}_n$, then the derandomization succeeds with precisely the same probability and over the same distribution. Further details appear in Section 6.

## 1.4 Catalytic computation towards an even smaller footprint

In the model of *catalytic computation*, introduced by Buhrman *et al.* [BCK+14], we enrich the space-bounded model with an *auxiliary memory*, that initially already stores some data. While we are allowed to use the auxiliary memory for *our* computation, in addition to the standard work tape, the auxiliary memory needs to be restored to its original content after use. Can such a seemingly restrictive usage be useful for computation? The work of [BCK+14] and followup works showed that it is indeed the case. Here, we give a possible *application* of catalytic space to derandomization, a connection that as far as we know, was not suggested before.

Suppose that our hard language from Assumption 4 can be computed catalytically, that is, most of the space used to compute it can be eventually restored. More concretely, consider the following assumption:

**Assumption 4.** *The language from Assumption 1 is computable in space $n$ using additional $C \cdot n$ auxiliary catalytic space.*

Then, we can show:

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

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

*for some fixed universal constant $c$.*

A similar result also holds in the non black-box setting.

Theorem 4 brings us tantalizingly close to derandomization without added memory footprint. Interestingly, the regime of parameters in Assumption 4, where the work space is only a small constant fraction of the catalytic space, has not been studied in the catalytic computation literature. (So far, the focus has been on the particular case where the catalytic space is exponential in the working space.) We thus view Assumption 3 also as a motivation to study catalytic computation in other regimes of parameters, which are useful for the study of derandomization.

# 2 Technical Overview

In Section 2.1 we describe the proof of Theorem 1, the main component of which (a new error-correcting code – see Theorem 2.1) will be used in the subsequent proofs. Then, in Section 2.2 we describe the proofs of Theorem 2 and Theorem 4. In particular, we show how to eliminate derandomization overheads, using a new algorithmic idea for derandomization, a particular type of cryptographic PRG, and an assumption about catalytic space. The proof of Theorem 3 is described in Section 2.3, and requires the strengthening of all the components described in Sections 2.1 and 2.2.

## 2.1 Warm-up: Hardness amplification for $\mathbf{TC}^0$ circuits in linear space

The proof of Theorem 1 relies on the standard hardness-vs.-randomness approach, following [NW94; IW97; STV01]: Given an input $x \in \{0,1\}^n$ the derandomization algorithm first computes the truth-table $f \in \{0,1\}^{\mathrm{poly}(n)}$ of the hard function (on input length $O(\log n)$); then it transforms $f$ into a truth-table $\bar{f} \in \{0,1\}^{\mathrm{poly}(|f|)}$ of a function that is hard on average, using a locally list-decodable error-correcting code; and finally it uses the Nisan–Wigderson generator to transform $\bar{f}$ into pseudorandom strings on which the probabilistic machine is evaluated with input $x$ (see, e.g., [Gol08, Chapters 7, 8], [AB09, Chapters 19, 20]).

The bottleneck in this approach is the worst-case to average-case reduction underlying the transformation of $f$ to $\bar{f}$. To prove that the derandomization works for logspace machines, it suffices for $\bar{f}$ to be hard on $1/2 - 2^{-\varepsilon \cdot m}$ fraction of its inputs for ROBPs of width $2^{\varepsilon \cdot m}$, for some $\varepsilon > 0$ and where $m = \log(|\bar{f}|)$.[6] In order to deduce this conclusion using the standard argument of [STV01], we need to assume that $f$ itself is hard (in the worst-case) for $\mathcal{C}$-procedures with oracle access to ROBPs of linear width, where $\mathcal{C}$ is the complexity of the local list-decoding algorithm of the code.

Loosely speaking, to decode from distance $1/2 - \delta$ (and deduce hardness on $1/2 - \delta$ fraction of the inputs), the procedure $\mathcal{C}$ needs to be able to compute the majority function (on $\Theta(1/\delta)$ bits, which in our setting would be $\Theta(2^{\varepsilon \cdot m})$ bits; see [GGH+07]).[7] Unfortunately, even when allowing $\mathcal{C} = \mathbf{TC}^0$, the best known decoder, from [GGH+07], only handles $\delta = 2^{-\sqrt{m}}$, which is too large for us. The codes that *are* typically used for hardness amplification with $\delta = 2^{-\varepsilon \cdot m}$ (i.e., the ones from [IW97; STV01]) are not known to be locally list-decodable in complexity as low as $\mathbf{TC}^0$.[8]

[6] This is actually an over-simplification, and what we actually need is for $\bar{f}$ to be hard on $1/2 - 2^{-\varepsilon \cdot m}$ fraction of the its inputs for $\mathbf{AC}^0$ circuits that have oracle access to an ROBPs of width $2^{\varepsilon \cdot m}$ (this follows from the standard reconstruction argument of [NW94]). In this high-level overview we ignore the $\mathbf{AC}^0$ overhead, for simplicity of presentation.

[7] In fact, a similar statement holds for any "black-box" worst-case to average-case hardness amplification [Vio03; SV08; GSV18].

[8] The bottleneck in both cases is local list-decoding of the Reed-Muller code; see [CT21a] for a recent construction of a decoder in logspace-uniform $\mathbf{NC}$.

The key observation allowing us to bridge this gap is that for our application of hardness amplification, we do not have to insist on the $\mathbf{TC}^0$ circuit being of size $\mathrm{poly}(\ell)$, where $\ell = \log(|f|)$, as in the standard setting of local coding. In fact, in our setting we can allow a circuit of size $2^{\varepsilon \cdot \ell}$. Given this relaxation, we construct the following suitable code.

**Theorem 2.1** (see also Theorem 4.2)**.** *There exists a universal constant $c > 1$ such that for any constant $\gamma \in (0,1)$ the following holds. For every $k \in \mathbb{N}$ and $\varepsilon > 0$, there exists a logspace-computable code $\mathcal{C} \colon \{0,1\}^k \to \{0,1\}^n$, for $n = (\frac{k}{\varepsilon})^{c/\gamma}$, that is locally list decodable from $1/2 + \varepsilon$ fraction of agreement by constant-depth threshold circuits of size $k^\gamma \cdot (1/\varepsilon)^c$.*

At a high level, the proof of Theorem 2.1 (wherein one should think of $k = 2^m$) combines three known code constructions:

1. A small modification of the code of [GGH+07], which uniquely decodes from agreement $1 - \frac{1}{25}$ using $\mathbf{TC}^0$ circuits;
2. the derandomized direct-product code of [IW97], which $(1 - \frac{1}{25})$-approximately list-decodes from agreement $\eta = 2^{-O(\varepsilon \cdot m)}$ using a $\mathbf{TC}^0$ circuit of size $2^{O(\varepsilon \cdot m)}$; and,
3. the Hadamard code, which we concatenate with the direct product code and is list-decodable from agreement $\frac{1}{2} + 2^{-\varepsilon \cdot m}$ by $\mathbf{TC}^0$ circuits of size $\mathrm{poly}(m)$.

As a corollary, we obtain a $\mathbf{TC}^0$-computable worst-case to average-case reduction for computing functions in $\mathbf{DSPACE}[O(n)]$; see Corollary 4.1. This reduction handles the "high-end" parameter regime, which previous reductions for functions in $\mathbf{DSPACE}[O(n)]$ did not handle (see [Spi96; GK08; GGH+07]), and is incomparable to reductions computable by probabilistic (uniform) algorithms [TV07; CRT22].

**The decoder's complexity.** For our results we crucially rely on the fact that the decoder can be implemented in $\mathbf{TC}^0$ (e.g., when deducing black-box derandomization from hardness for $\mathbf{TC}^0$ circuits with oracle access to branching programs, or when deducing non-black-box derandomization). We suspect that it is possible to construct a code with weaker guarantees – namely, a logspace decoder, rather than a $\mathbf{TC}^0$ decoder – using simpler techniques (i.e., replacing the "outer" code of [GGH+07] by more classical tools).

## 2.2 Derandomization with minimal footprint using PRGs

Let $S = C \cdot \log n$ denote the space complexity of the machine $M$ we wish to derandomize (the result for arbitrary $S$ will follow from padding). At a high-level, our construction follows an approach first introduced in [CT21b], which composes two "low-cost" PRGs:

- An inner PRG that stretches $O(\log n)$ bits to $n^\eta$ bits for some tiny constant $\eta > 0$, and,
- an outer PRG that stretches $n^\eta$ bits to $n^C$ bits.

Specifically, as in [CT21a], we take the inner PRG, denoted by $\mathsf{NW}$, to be an appropriately parameterized Nisan-Wigderson PRG [NW94] with a hard truth-table $f \in \{0,1\}^{n^2}$, and the outer PRG, denoted by $G^{\mathsf{cry}}$, to be one that relies on a cryptographic assumption.

Unfortunately, materializing this approach in the current setting turns out to be significantly more subtle than in [CT21b]. To see this, observe that the final computation iterates over seeds $s \in \{0,1\}^{O(\log n)}$, and for each $s$ computes

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

where $f$ is the truth-table of the hypothesized hard function. To compute this using space-efficient composition, we use the following chain of simulations:

1. Simulate $M(x,\cdot)$, and whenever it queries its second input –
2. Simulate $G^{\mathsf{cry}}$, and whenever it queries its input –
3. Simulate $\mathsf{NW}^f(s)$, and whenever it queries $f$ –
4. Compute the corresponding bit of $f$.

Recall that, using space-efficient (emulative) composition, the complexity of the final construction is additive in the space complexity of each of its components, plus additional overheads that are logarithmic in the output length of each component. (The latter logarithmic overhead is caused by the fact that we are simulating a virtual input head for each component. See Proposition 3.2.) A naive implementation of this approach yields space complexity of $3S + \mathsf{Space}(G^{\mathsf{cry}}) + \mathsf{Space}(\mathsf{NW})$, where $\mathsf{Space}(\cdot)$ denotes the space complexity of the corresponding algorithm, and we ignore factors of the form $c \cdot \log n$ where $c > 1$ is a universal constant that doesn't depend on $S$.

**A more efficient derandomization.** Our first observation is that the standard way of derandomizing probabilistic space-$S$ machine is wasteful. There, we think of the probabilistic machine as reading a tape of random bits, sequentially; and when simulating it deterministically, we keep track of a counter $i \in [2^S]$, and whenever the machine wishes to read a random bit, we answer using the $i$-th bit in the pseudorandom output of the generator, and update $i \leftarrow i + 1$.

It might (mistakenly) seem that using a dedicated counter is necessary, because we must ensure that the machine reads each bit in the random (or pseudorandom) sequence exactly once. However, this intuition turns out to be false: Instead of keeping a dedicated counter $i \in [2^S]$, *we can use the machine's own configuration as a counter*. Specifically, recall that at each step the machine has some configuration $\sigma \in \{0,1\}^S$ describing the contents of its work tapes, its current state, and the locations of its heads.[9] Moreover, since for every input $x$ and fixed sequence $r$ of coins, the execution of $M(x,r)$ halts, any configuration $\sigma \in \{0,1\}^S$ is encountered *at most once* during the execution of $M$ (see Claim 3.3).

[9] Indeed, we count the location of the heads and the state in the configuration of the machine, and in fact assume that they are written on dedicated worktapes; see Section 3 for the precise details.

Thus, we consider the following machine $\bar{M}$, which simulates $M$ using oracle access to a sequence of random coins but without the overhead of keeping a counter:

> Simulate $M$, and whenever $M$ tries to flip a random coin, access the sequence of random coins at location $\sigma$, where $\sigma$ is $M$'s current configuration.

Since the functionality of $\bar{M}$ and of $M$ at any input $x$, with uniform coins, is identical, it suffices to faithfully simulate $\bar{M}$ with pseudorandom coins.

At this point the space complexity of the derandomization is essentially

$$2S + \mathsf{Space}(G^{\mathsf{cry}}) + \mathsf{Space}(\mathsf{NW}).$$

Since NW maps a truth-table $f$ of length $n^2$ to pseudorandom strings of length $n^\eta$, it can be computed in space $c' \cdot \log n$ for a universal $c' > 1$ (see Section 5.1). Thus, our last step is to bound the space complexity of $G^{\mathsf{cry}}$.

We do so by relying on a specific PRG whose space complexity is logarithmic in its input length $n^\eta$ and sub-logarithmic in its output length $n^C$. A natural candidate for such a PRG arises from the "cryptography in $\mathbf{NC}^0$" literature (see, e.g., [AIK06; IKO+08; App14; RS21]), and in particular we can use Goldreich's PRG [Gol11b]. The latter PRG relies on a bipartite lossless expander $\Gamma\colon [n^C] \times [d] \to [n^\eta]$ with a small left-degree $d$, and on a predicate $P\colon \{0,1\}^d \to \{0,1\}$. For $s \in \{0,1\}^{n^\eta}$ and $i \in [n^C]$, the $i$-th output of $G^{\mathsf{cry}}(s)$ is

$$P\left(s_{\Gamma(i,1)}, ..., s_{\Gamma(i,d)}\right),$$

where $\Gamma(i,1), \ldots, \Gamma(i,d)$ are the $d$ neighbors of $i$ in the expander.

In the cryptography literature, the graph is often taken to be a random one, but in our setting we need a lossless expander whose neighbor function is computable in space $c'' \cdot \log n$, and also a predicate known to withstand existing attacks that is computable in space $c'' \cdot \log n$, where in both cases $c'' > 1$ is a universal constant. We use the recent expander construction of Kalev and Ta-Shma [KT22], whose degree is $d = \mathrm{polylog}(n)$ and whose neighbor function is computable in space $O(\mathrm{loglog}n)$, and a predicate introduced by Applebaum and Raykov [AR16] that is computable in space $O(\log d) = O(\mathrm{loglog}n)$. See Section 5.3.1 for further details.

This brings the complexity of the deterministic simulation of $M$ on each particular seed to be $2S + c \cdot \log n$ for some universal constant $c$, and after enumerating over all seeds and taking the majority output, the space complexity increases only by an additive factor of, say, at most $c \cdot \log n$.

**The reconstruction argument.** We prove that the derandomization works using a reconstruction argument. Specifically, in the derandomization, we instantiate the NW generator with the code from Theorem 2.1, and rely on the reconstruction argument of NW and on the local list-decoding algorithm of the code to transform any ROBP distinguisher for the PRG (which arises from the computation of the space-$S$ machine $M$ on a fixed input $x$) into an efficient procedure that computes $f$.

The details of the reconstruction procedure appear in Theorem 5.1, so let us only highlight the important points in the argument. First, our distinguisher is actually derived from $\bar{M}$, the machine that reads bits according to its configuration. Secondly, by a standard analysis of PRG composition, the distinguisher for $\mathsf{NW}^f$ is not just the ROBP derived from $\bar{M}$ and $x$, but actually the composed procedure

$$D(r) = \bar{M}^{G^{\mathsf{cry}}(r)}(x).$$

This increases the complexity of the distinguisher from a simple ROBP to a bounded-space machine. Lastly, while the machine implementing $D$ uses space at least $S = C \cdot \log n$, the amount of non-uniform advice that it uses is much smaller than $2^S = n^C$. Specifically, it uses only $|x| = n = 2^{\log(|f|)/2}$ bits of advice.

To sum up, if the derandomization fails on some input $x$, then there exists a $\mathbf{TC}^0$ circuit $C$ of size $n^\varepsilon$, and a function $D$ that is computable in space $\approx C \cdot \log n$ with $n$ bits of advice,[10] such that $C^D(i) = f_i$ for all $i \in [n^2]$. Finally, using the fact that $C$ is a $\mathbf{TC}^0$ circuit, we show that $C^D$ itself can be computed by a TM with advice, whose space complexity is only slightly larger than the space complexity of $D$. This contradicts the hardness of $f$.

**Obtaining a sub-double space overhead.** The derandomization above takes $2S + c \cdot \log n$ space, where the increase from $S$ to $2S$ is caused by the space complexity of computing $f$. Indeed, it seems unavoidable that we will need space larger than $S$ to compute $f$, because we are assuming that it is hard for algorithms running in space $S$.

The key observation to reducing this overhead is that if $f$ is computable in *catalytic* space $S$, we can roughly use the existing used worktape cells – which, at any point in time, contain the current configuration of the derandomized machine – in order to compute each query of $\mathsf{NW}$ to $f$. Specifically, whenever the derandomization machine queries $f$ at location $i \in [n^2]$, we compute $f_i$ using (mostly) the existing space in a catalytic way. After the computation of the bit $f_i$ ends, the original content of the worktapes is restored. The key point is that since the configuration of the machine does not change during the computation of $i \mapsto f_i$, nor is this computation dependent on the configuration in any way, the correctness of the procedure is maintained.

Thus, under this strengthened hypothesis that $f$ is computable in small catalytic space, the final space complexity of the derandomization algorithm is just $S + c \cdot \log n$.

**An alternative to Assumption 1.** Recall that $D(r)$ above can be computed by a bounded-space machine that uses $|x|$ bits of advice. While indeed $D$ is not a read-*once* branching program, the computation of $D(r)$ is *oblivious*. Namely, computing $C^{\mathsf{cry}}$ can be done by a bounded-width branching program that at each layer queries several locations of $r$, however these locations are determined only by the underlying expander $\Gamma$. Thus, we can model $C^D$ as a $\mathbf{TC}^0$ circuit with (non-adaptive) oracle access to branching programs of

[10] The precise complexity of $D$ is $(C + 1 + \varepsilon) \cdot \log n$, but in the high-level overview we ignore these minor overheads, for simplicity of presentation.

the aforementioned type. Moreover, we will later see that the $\mathbf{TC}^0$ circuit can be space-efficiently generated using a short advice. The formal assumption is given in Assumption 5.10, and can replace Assumption 1 for both the double and sub-double overhead results.

## 2.3 Non black-box derandomization with minimal memory footprint

Set $S = C \cdot \log n$ and recall that we wish to obtain the same conclusions for $\mathbf{BPSPACE}[S]$ as in Section 2.2, but from different assumptions. Specifically, we assume the existence of a function $f\colon \{0,1\}^n \to \{0,1\}^{n^2}$ computable in space $S' = S + O(\log n)$ such that for every probabilistic algorithm $R$ running in space $S_R = S' + O(\log n)$, and every $x \in \{0,1\}^n$, the algorithm $R(x)$ fails to print a *compressed* version of $f(x)$ (except with small probability). In this context, a compressed version means a Turing machine of description size $O(n) = O(\sqrt{|f(x)|})$ that runs in space roughly $S + \log n < S'$.

Following ideas from [CT21a; LP22a], we will construct a *targeted* PRG, which is an algorithm that maps any input $x$ to a set of pseudorandom strings that will fool the machine $M$ with this particular input $x$. As in those previous works, our targeted PRG is based on the Nisan–Wigderson generator, and we analyze it using an *instance-wise* hardness vs. randomness tradeoff. Specifically, we show that if the derandomization fails on an input $x$, then a probabilistic space-$S_R$ machine $R$ succeeds in mapping the same fixed $x$ to a compressed version of $f(x)$. This yields Theorem 3, and also the more general version mentioned after the theorem's statement: For every distribution $\mathbf{x}$ over the inputs, if the probability over $x \sim \mathbf{x}$ that $R$ fails to print compressed version of $f(x)$ is $1 - \mu$, then the derandomization succeeds on $1 - \mu$ of the inputs $x \sim \mathbf{x}$.

The derandomization itself is similar to the one from Section 2.2, with a minor difference that is nevertheless crucial. Instead of instantiating $\mathsf{NW}$ with $f$ that is the truth-table of a hypothesized hard function, we instantiate $\mathsf{NW}$ with $f = f(x)$ obtained from the input $x$. That is, we compute the majority, over seeds $s \in \{0,1\}^{O(\log n)}$, of

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

Note that the complexity of the derandomization algorithm is essentially identical to that of the algorithm from Section 2.2. Thus, the only question is – why does it work?

**Analysis.** The main argument underlying Theorem 3 is proving that there exists a space-$S_R$ algorithm $R = R_M$ satisfying the following: For any fixed $x \in \{0,1\}^n$, when $\mathsf{NW}$ is instantiated with the code from Theorem 2.1, if $\mathsf{NW}^{f(x)}$ does not fool $M$, then $R(x)$ prints a compressed version of $f(x)$.

The intuition for why this might be possible dates back to [IW01], who showed that the reconstruction algorithm of $\mathsf{NW}$, which maps a distinguisher to a small circuit for the hard truth-table, can be made *uniform* – as long as it is allowed to make queries to the hard truth-table. Recall that in our setting, the algorithm $R$ explicitly gets the input $x$, and we are allowing $R$ to run in space that is slightly larger than the space complexity of

computing $f(x)$. Therefore, $R$ can simulate the reconstruction algorithm, and whenever the latter queries an index $i$ of $f(x)$, the algorithm $R$ simply computes $f(x)$ and returns the relevant bit.

The main technical challenge that we are faced with is making the algorithm $R$ not only uniform, but also a *small-space algorithm*, and doing so when the underlying *code for hardness amplification* is the one from Theorem 2.1. The resulting statement appears in Theorem 6.1, and its proof is the most technically subtle argument in this paper. In the rest of the section we describe the proof, at a high-level.

**Low-space uniform reconstruction and decoding.** We first strengthen the analysis of the code $\mathcal{C}$ from Theorem 2.1, to show that not only is it locally list-decodable, but that it also has a *probabilistic space-$O(\log n)$ uniform decoder, which does not need non-uniform advice* (but rather uses queries to the corrupt codeword). The algorithm $R$ will answer this decoder's queries to the corrupt codeword $\mathcal{C}(f(x))$ by computing $f(x)$ and then $\mathcal{C}$, which it can do in its allotted space. We compose this uniform decoder for $\mathcal{C}$ with a space-$O(\log n)$ reconstruction algorithm for NW.

We stress that the two algorithms underlying $R$ – the decoder, and the NW reconstruction – run in space $O(\log n)$, but print a procedure of description size $n^{\Omega(1)}$. Thus, not only do the two algorithms need to print a description of a procedure without remembering most of the functionality of the machine that they printed so far – but also the algorithms cannot even *evaluate* the procedures that they print.

The key observation is that in both cases, the decoding/reconstruction prints a procedure *almost all of which is a large, static, truth-table*. To see this, let us focus for simplicity on the NW reconstruction algorithm.[11] Recall that this algorithm implements very simple functionality, which can be described by a logspace-uniform constant-depth circuit of size $\mathrm{polylog}(n)$, and that is hard-wired with "static" information of size $n^{\varepsilon}$ that is mostly obtained from queries to $\mathcal{C}(f(x))$.[12] With some low-level care, we can design an algorithm that queries $\mathcal{C}(f(x))$ and prints a machine that implements that functionality, and has states encoding the foregoing static information. Thus, the algorithm which prints the machine does not need to remember static information that is already printed.

A related complication arises because both the decoding algorithm of $\mathcal{C}$ and the reconstruction algorithm of NW actually succeed only with small probability. The standard approach (e.g., in [IW01; CT21a; LP22a]) is for $R$ to use queries to $\mathcal{C}(f(x))$ to estimate the agreement of the procedure that it outputs with $\mathcal{C}(f(x))$, repeating the experiment until it gets a procedure with sufficiently large agreement. Since in our case $R$ cannot evaluate the procedure that it prints, it cannot take this approach. Instead, $R$ prints a procedure that performs this "success amplification" functionality by itself. We leave the details to the technical section.

[11] The computational bottleneck in the decoder for $\mathcal{C}$ is the decoder for the derandomized direct product code of [IW97], which acts in a similar way to the reconstruction of NW. Thus, we use similar ideas to handle both the reconstruction of NW and the decoder of $\mathcal{C}$.

[12] The information consists of an index $i$ (used for a hybrid argument), of a combinatorial design, of values for the seed outside the $i$-th set in the design, and of $n^{\eta}$ partial truth-tables.

**Composing the two algorithms.** The description above refers vaguely to "the procedure" that $R$ print, and being more accurate, this procedure is a $\mathbf{TC}^0$ circuit $C$ of size $n^\varepsilon$ making queries to a space-$S$ machine $D$ that uses $n$ bits of advice. This is not enough, since our goal is for $R$ to print a single Turing machine of description size $O(\sqrt{|f(x)|}) = O(n)$ running in space $S + \log n < S'$.

Bridging this gap requires more care in composing the two algorithms. Specifically, our algorithm $R$ prints a machine whose states encode the circuit $C$, and that implements the standard DFS-style emulation of $\mathbf{NC}^1 \supseteq \mathbf{TC}^0$ circuits in logspace, while reading the description of the hard-coded $C$ out of its own states. The space overhead of the emulation itself is $O(\log |C|) = O(\log(n^\varepsilon))$, and the machine also needs to compute the values of the gates along each DFS path. In particular, this means that we need to ensure that each path contains at most one oracle call to $D$ (otherwise the machine's space complexity will be larger than $2S$). For this purpose, in our strengthened analysis of $\mathcal{C}$ we ensure that its decoding procedure only makes *non-adaptive queries*. This allows us to bound the space complexity of the machine that $R$ prints by $S + O(\varepsilon \cdot \log n) \leq S + \log n$, as desired.

# 3 Preliminaries

## 3.1 Space-bounded computation

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 $\mathbf{DSPACE}(s(n))$ 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.[13] 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 $1/3$.

[13] 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 $\mathbf{BPSPACE}(s(n))$ 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 3.1.** *Let $M$ be a (deterministic or probabilistic) space-bounded TM with $C > 1$ work tapes, such that on input of length $n$ uses space $s(n) \geq \log n$ (in total over all its work tapes). Then, $M$ can be simulated by a TM with a single work tape that uses $s(n) + O(C \cdot \log(s(n)))$ space.*

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

**Proposition 3.2** ([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 3.2 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).

**Configurations of space-bounded machines.** A space-$s$ machine (let it be deterministic or probabilistic) has $2^{s'}$ possible configurations, where $s' = s + O(\log s)$. Given a TM $M$, it is possible to convert it into another machine $M'$ that prints its (original) configuration on a dedicated worktape, paying at most a constant factor blowup in the number of states. (This conversion is explicit.)

Given a probabilistic TM $M(x, r)$, instead of reading the bits of $r$ in the standard unidirectional manner, we can think of an oracle TM $\bar{M}^{r'}(x)$ that simulates $M$, and whenever $M$ reads a random bit, $\bar{M}$ queries $r'$ in location $q$, where $q$ denotes $M$'s corresponding configuration. The key point is that while the bits of $r'$ are read out of order, each one will only be read at most once. This follows from the fact that $M$ itself always halts (within $2^{s'}$ steps).

**Claim 3.3.** *Let $M$ and $\bar{M}$ as above. Then, for sufficiently long independent and uniform $\mathbf{r}, \mathbf{r}'$, and any $x$, it holds that* $\Pr[M(x,\mathbf{r})=1] = \Pr[\bar{M}^{\mathbf{r}'}(x)=1]$.

*Proof Sketch.* It suffices to show that for each $r'$, $\bar{M}(x)$ reads each bit of $r'$ at most once. This follows from the fact that $\bar{M}$ never repeats a configuration. Indeed, suppose that it visits some configuration $c$ twice upon reading $x$ with oracle access to some $r'$. Thus, since $\bar{M}$ will then read the same bits in $r'$, $\bar{M}^{r'}(x)$ will never halt. This means that there *exists* some string $r$ (determined by $r'$) on which $M(x,r)$ will never halt, but $M$ halts on every input and randomness string. ■

Throughout the paper, we will freely use a description of TMs as strings. We thus fix any canonical standard way of doing so (i.e., by fixing some universal Turing machine).

## 3.2 Additional complexity classes, and branching programs

We will use the standard definitions of circuit classes. In particular, an $\mathbf{AC}^i$ circuit is a Boolean circuit with depth $O(\log^i n)$ over the De Morgan basis with unbounded fan-in gates. In $\mathbf{TC}^i$, we allow Majority gates in addition to NOT, OR, and AND. 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 $O(w \log s)$.

**Read-once branching programs.** We use the standard definition of layered read-once branching programs. For a length parameter $n \in \mathbb{N}$, a width parameter $w \in \mathbb{N}$, and an alphabet $\Sigma$, an $[n,w]_\Sigma$ ROBP $B$ is specified by an initial state $v_0 \in [w]$, a set of accepting states $V_{\mathsf{acc}} \subseteq [w]$ and a sequence of transition functions $B_i \colon [w] \times \Sigma \to [w]$ for $i \in [n]$. The ROBP naturally defines a function $B \colon \Sigma^n \to \{0,1\}$: Start at $v_0$, 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 $x$ if $v_n \in V_{\mathsf{acc}}$, and rejects otherwise.

Next, we define constant-depth circuits with oracle access to functions computable by bounded-width ROBPs.

**Definition 3.4.** *We say that a Boolean circuit $C$ is an $\mathbf{TC}^0_{\mathsf{robp}}$ circuit is there exists an $[M,M]_{\{0,1\}}$ ROBP $B$, for $M \leq \mathsf{size}(C)$, such that $C$ is a $\mathbf{TC}^0$ circuit with oracle calls to $B$.*

**Catalytic computation.** Catalytic computation, defined by Buhrman *et al.* [BCK+14] (see also [Kou+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 machine with an *auxiliary tape*. For every possible initial setting of the auxiliary 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 accept or reject.

In the catalytic computation literature, the common setting is $s_A = 2^{O(s)}$. We will work with a more fine-grained separation, and consider the case when catalytic computation offers an advantage over standard space-bounded computation for $s_A$ being only slightly larger than $s$.

## 3.3 Error-correcting codes

We say that an error correcting code $\mathcal{C} \colon \Sigma^k \to \Sigma^n$ has *relative distance* $\delta$ 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 $(\rho, L)$ 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 $\rho$ 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 3.5** (locally approximately list-decodable code)**.** *We say that a code $\mathcal{C} \colon \Sigma^k \to \Sigma^n$ is $(\rho \to 1 - \delta, Q, L, \xi)$* locally approximately list decodable *by circuits of size $s$ if there exist randomized circuits $\mathsf{Dec}_1, \ldots, \mathsf{Dec}_L$, each of size $s$, that satisfy the following.*

- *Each $\mathsf{Dec}_i$ 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 $\rho$-fraction of its coordinates, there exists $j \in [L]$ such that*

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

*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.*

We note that when we do not pose any uniformity constraints, the output list size parameter $L$ may only implicit, in the sense that each $\mathsf{Dec}_i$ 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 $\mathsf{Dec}_i$-s, by standard error reduction, we can take $\xi = 1$ and incur only a minor loss in parameters. This leads us to the following, shorter definition:

**Definition 3.6** (locally approximately list-decodable code)**.** *We say that a code $\mathcal{C}\colon \Sigma^k \to \Sigma^n$ is $(\rho \to 1-\delta, Q)$* locally approximate list decodable *by circuits of size $s$ if there exist circuits $\mathsf{Dec}_1, \ldots, \mathsf{Dec}_L$ for some $L \leq 2^s$, each of size, that satisfy the following. For every $r \in \Sigma^n$, and $c = \mathcal{C}(x)$ that agrees with $r$ in at least $\rho$-fraction of its coordinates, there exists $j \in [L]$ such that*

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

*When $\delta = 0$, we say that $\mathcal{C}$ is locally list decodable, and use the notation $(\rho, Q)$. When $L = 1$, we say that $\mathcal{C}$ is locally (approximately) uniquely decodable.*

## 3.4 Samplers

**Definition 3.7** (strong sampler)**.** *A function $\mathsf{Samp}\colon \{0,1\}^m \times [k] \to \{0,1\}^n$ is a* strong $(\eta, \mu)$ (oblivious) sampler *if for any $H_1, \ldots, H_k \subseteq \{0,1\}^n$ it holds that*

$$\Pr_{x\in\{0,1\}^m}\left[\left|\Pr_{i\in[k]}\left[\mathsf{Samp}(x,i) \in H_i\right] - \mathop{\mathbb{E}}_{i\in[k]}\left[\rho(H_i)\right]\right| \leq \eta\right] \geq 1-\mu,$$

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

Random walks on expanders give strong samplers, with the following parameters.

**Theorem 3.8** ([Hea08, Theorem 1.3])**.** *For every $n \in \mathbb{N}$ and any $\eta, \mu > 0$ there exists an explicit strong $(\eta, \mu)$ sampler $\mathsf{Samp}\colon \{0,1\}^m \times [k] \to \{0,1\}^n$ where $k = O(\log(1/\mu)/\eta^2)$ and $m = n + O(k)$ that is computable in linear space. That is, given $x \in \{0,1\}^m$ and $y \in [k]$, $\mathsf{Samp}(x,y)$ is computable in space $O(m)$.*

We will also use the following strong sampler, that has better randomness complexity at the expense of worse sampling complexity.[14]

**Theorem 3.9** ([Gol11a; CL20])**.** *For every $n \in \mathbb{N}$ and any $\eta, \mu > 0$ there exists an explicit strong $(\eta, \mu)$ sampler $\mathsf{Samp}\colon \{0,1\}^m \times [k] \to \{0,1\}^n$ where $k = \mathrm{poly}(\log(1/\mu), 1/\eta)$ and $m = n + O(\log(1/(\eta\mu)))$ that is computable in linear space. That is, given $x \in \{0,1\}^m$ and $y \in [k]$, $\mathsf{Samp}(x,y)$ is computable in space $O(m)$.*

[14] We note that the "strongness" property does not appear in [RVW02; Gol11a; CL20] (the standard, non-strong, definition assumes $H_1 = \ldots = H_k$). 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]).

### 3.5 Combinatorial designs

**Definition 3.10** (combinatorial design)**.** *A family of sets $S_1, ..., S_k \subseteq [d]$ is called an* $(n, a)$-design *if each of the sets is of size $|S_i| = n$, and any two distinct sets $S_i, S_j$ satisfy $|S_i \cap S_j| \leq a$. The corresponding* design function $\mathsf{Des}\colon \{0,1\}^d \times [k] \to \{0,1\}^n$ *takes an input $z \in \{0,1\}^d$ and an index $i \in [k]$ and outputs the projection of $z$ to the coordinates in $S_i$.*

We will make use of logspace computable designs.[15]

**Theorem 3.11** ([KM02, Lemma 5.19])**.** *There is a universal constant $\bar{c} \geq 1$ such that for any constant $\alpha \in (0,1)$ the following holds for every sufficiently large $n \in \mathbb{N}$. There exists an algorithm that outputs an $(n, \alpha \cdot n)$-design $S_1, ..., S_k \subseteq [d]$, where $k = \left\lceil 2^{(\alpha/\bar{c}) \cdot n} \right\rceil$, and $d \leq (\bar{c}/\alpha) \cdot n$. On input $i \in [k]$, the algorithm runs in space $O(n)$ and outputs $S_i$. In particular, the corresponding design function $\mathsf{Des}\colon \{0,1\}^d \times [k] \to \{0,1\}^n$ is computable in space $O(n)$.*

# 4 Hardness Amplification for Space-Bounded Computation

Our goal in this section is to prove the following hardness amplification result: If there exists a language $L \in \mathbf{DSPACE}[O(n)]$ that is hard for a class $\mathcal{C}$ of non-uniform circuits in the worst-case, then there exists another language $L' \in \mathbf{DSPACE}[O(n)]$ that is hard for a related circuits class $\mathcal{C}'$ on $\frac{1}{2} + 2^{-\Omega(n)}$ fraction of the $n$-bit inputs. In the transformation of $f$ into $\bar{f}$, we want to maintain the complexity of the hard function, while minimizing the difference between $\mathcal{C}$ and $\mathcal{C}'$. In particular, we want $\mathcal{C}$ to consist of constant-depth threshold circuits of size $2^{\gamma \cdot n}$ with oracle access to $\mathcal{C}'$, for an arbitrarily small constant $\gamma > 0$. More formally:

**Corollary 4.1** (hardness amplification in linear space)**.** *Let $\mathcal{F}$ be a class of Boolean functions. Assume that for some constant $\varepsilon > 0$ there exists $L \in \mathbf{DSPACE}[O(n)]$ that is hard on all but finitely many input lengths for $\mathbf{TC}^0$ circuits of size $2^{\varepsilon \cdot n}$ with oracle access to $\mathcal{F}$.[16] Then, there exists $L' \in \mathbf{DSPACE}[O(n)]$ such that for every $f \in \mathcal{F}$ and sufficiently large $n \in \mathbb{N}$, it holds that $\Pr_{x \in \{0,1\}^n}[f(x) = L'(x)] \leq \frac{1}{2} + 2^{-\delta \cdot n}$, where $\delta = \delta(\varepsilon) > 0$.*

Following the (by now standard) idea introduced in [STV01], we use locally list decodable codes for hardness amplification, treating the local list decoding algorithm as a worst-case to average-case hardness reduction. Specifically, the main technical tool will be the construction of the following locally list-decodable code.

[15] The [KM02] result only guarantees the existence of designs for infinitely many $n$-s (more formally, for each $n$ there exists some $n' = O(n)$ for which the statement holds). This issue is resolved in [CT21a], following [HR03].

[16] That is, for every $f \in \mathcal{F}$ and every family $C$ of $\mathbf{TC}^0$ oracle circuits of size $2^{\varepsilon \cdot n}$ and every sufficiently large $n \in \mathbb{N}$ there is $x \in \{0,1\}^n$ such that $C^f(x) \neq L(x)$.

**Theorem 4.2** (locally list-decodable code)**.** *There exists a universal constant $c > 1$ such that for any constant $\gamma \in (0,1)$ the following holds. For every $k \in \mathbb{N}$ and $\varepsilon > 0$, there exists a logspace-computable code $\mathcal{C}\colon \{0,1\}^k \to \{0,1\}^n$, for $n = (\frac{k}{\varepsilon})^{c/\gamma}$, that is*

$$\left(\rho = \frac{1}{2} + \varepsilon, Q = \left(\frac{\log k}{\varepsilon}\right)^c\right)$$

*locally list decodable $\mathbf{TC}^0$ circuits of size $k^\gamma \cdot (1/\varepsilon)^c$ that make non-adaptive queries.*

Most of this section will be devoted to the proof of Theorem 4.2. Before turning to the proof itself, let us state the worst-case to average-case reduction that follows from the local decoding algorithm (following [STV01]).

**Proposition 4.3** (hardness amplification in linear space)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $f\colon \{0,1\}^\ell \to \{0,1\}$, and let $\gamma \in (0,1)$. For $k = 2^\ell$, identify $f \in \{0,1\}^k$ with the truth table of $f$, and let $\mathcal{C}\colon \{0,1\}^k \to \{0,1\}^{k'}$ be the code from Theorem 4.2 instantiated with parameter $\gamma$ and to be locally list decode from agreement $\frac{1}{2} + \varepsilon$ for some $\varepsilon > 0$. Then, for any function $\mathcal{O}\colon \{0,1\}^{\ell'} \to \{0,1\}$ satisfying*

$$\Pr_{x \in \{0,1\}^{\ell'}}[\mathcal{O}(x) = \mathcal{C}(f)(x)] \geq \frac{1}{2} + \varepsilon,$$

*there exists a $\mathbf{TC}^0$ oracle circuit $\mathcal{A}\colon \{0,1\}^\ell \to \{0,1\}$ of depth $c$ and size $2^{\gamma \cdot \ell}/\varepsilon^c$ such that $\mathcal{A}^{\mathcal{O}}$ computes $f$ exactly and makes at most $(\ell/\varepsilon)^c$ oracle queries.*

Corollary 4.1 can now be easily established.

*Proof of Corollary 4.1.* Let $f$ be the truth table of $L$ on inputs of length $\ell$, and set $k = 2^\ell$, $\gamma = 2\varepsilon$, and $\delta = \varepsilon/c$ where $c$ is the constant from Proposition 4.3. Let $\mathcal{C}\colon \{0,1\}^k \to \{0,1\}^{k'}$ be the code from Theorem 4.2 set with agreement parameter $\frac{1}{2} + 2^{-\delta \cdot \ell}$. The language $L'$ on inputs of length $\ell' = \log k'$ is determined by the truth table $\mathcal{C}(f_\ell)$.[17] The function $\mathcal{C}$ is computable in space $O(\log k) = O(\ell)$, so indeed $f'$ is computable in linear space, and $L' \in \mathbf{DSPACE}[O(n)]$.

Next, fix some large enough $\ell' \in \mathbb{N}$ and assume towards a contradiction that there exists $f' \in \mathcal{F}$ on inputs of length $\ell'$ such that $\Pr_{x \in \{0,1\}^{\ell'}}[f'(x) = L'(x)] > \frac{1}{2} + 2^{-\delta \cdot \ell}$. By our construction, $L'(x) = \mathcal{C}(f)(x)$ for some $f\colon \{0,1\}^\ell \to \{0,1\}$. Then, Proposition 4.3 tells us that there exists a $\mathbf{TC}^0$ oracle circuit $\mathcal{A}\colon \{0,1\}^\ell \to \{0,1\}$ of size $2^{\gamma \cdot \ell}/2^{-c\delta\ell} = 2^{\varepsilon \cdot \ell}$ so that $\mathcal{A}^{f'}$ computes $f$ exactly. This contradicts our hardness assumption on $L$. ■

## 4.1 Known code constructions

The code underlying Theorem 4.2 will be a composition of three different (known) error-correcting codes. The first one is the code of Goldwasser *et al.* [GGH+07], which is locally *uniquely* decodable from a $(1/25)$-fraction of errors by $\mathbf{TC}^0$ circuits. That is:

[17] We can extend $L'$ to all large enough inputs lengths while retaining its properties given in the proof. We omit the details.

**Theorem 4.4** (locally uniquely decodable code with an $\mathbf{TC}^0$ decoder [GGH+07])**.** *There exists a logspace-computable code*

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

*where* $k' = \mathrm{poly}(k)$*, such that* $\mathsf{GGHKR}'$ *is locally unique decodable from* $\rho = \frac{24}{25}$ *fraction of agreement by* $\mathbf{TC}^0$ *circuits of size* $2^{\mathrm{polyloglog}(k)}$ *making non-adaptive queries.*

The fact that $\mathsf{GGHKR}'$ is computable in logspace is not explicit in [GGH+07], so we give the proof in Appendix B.1. Moreover, the code underlying Theorem 4.4 differs slightly from the original construction in [GGH+07] (which is why we denote it by $\mathsf{GGHKR}'$ rather than just GGHKR), and moreover they prove that it's decodable in (adaptive) $\mathbf{AC}^0$ of size $\mathrm{polylog}(k)$ whereas we prove that it's decodable in *non-adaptive* $\mathbf{TC}^0$ of size $2^{\mathrm{polyloglog}(k)}$; we elaborate on the construction and the difference in Appendices B.1 and B.2.

The next ingredient is the derandomized direct-product code of Impagliazzo and Wigderson [IW97], which is locally *approximately list-decodable*. Specifically, the local list decoding algorithm of this code is a $\mathbf{TC}^0$ circuit, and it takes a messages that agrees with a codeword on $\varepsilon > 0$ of the entries, and recovers a message that agrees with the codeword on $1 - \delta$ of the entries (we will use this with $\delta = 1/25$, to compose this code with the one in Theorem 4.4).

**Theorem 4.5** (locally approximately list-decodable code with a $\mathbf{TC}^0$ decoder [IW97])**.** *There exists a constant* $c_{\mathsf{IW}} > 1$ *such that for any two constants* $\delta_{\mathsf{IW}}, \gamma_{\mathsf{IW}} > 0$*, and every space-computable* $\varepsilon_{\mathsf{IW}} \colon \mathbb{N} \to (0,1)$*, the following holds. There exists a logspace-computable code*

$$\mathsf{IW} \colon \{0,1\}^{k'} \to (\{0,1\}^r)^{k''} ,$$

*where* $k'' = (k'/\varepsilon_{\mathsf{IW}})^{c_{\mathsf{IW}} \cdot (1/\gamma_{\mathsf{IW}} + 1/{\delta_{\mathsf{IW}}}^2)}$ *and* $r = (c_{\mathsf{IW}}/{\delta_{\mathsf{IW}}}^2) \cdot \log(1/\varepsilon_{\mathsf{IW}})$*, such that* IW *is*

$$\left( \varepsilon_{\mathsf{IW}} \to 1 - \delta_{\mathsf{IW}}, Q = \left( \frac{\log k'}{{\varepsilon_{\mathsf{IW}}}^2} \right)^{c_{\mathsf{IW}}} \right)$$

*locally approximately list-decodable by constant-depth oracle circuits of size* $(k')^{\gamma_{\mathsf{IW}}}/{\varepsilon_{\mathsf{IW}}}^2$*. Each decoding circuit has one majority gate of fan-in at most* $Q$*, and makes at most* $Q$ *non-adaptive oracle queries.*

The statement above does not explicitly appear in [IW97], but their proof already supports this statement. For completeness, we include a full proof in Appendix A.

The last code is simply the Hadamard code, whose list-decodability is established by the Goldreich-Levin algorithm [GL89].

**Theorem 4.6** (list-decodable code [GL89])**.** *For every* $r \in \mathbb{N}$ *and any* $\varepsilon < \frac{1}{2}$*, the Hadamard code*

$$\mathsf{Had} \colon \{0,1\}^r \to \{0,1\}^{2^r}$$

*is* $(\rho = \frac{1}{2} + \varepsilon, Q = O(1/\varepsilon^2), L = O(1/\varepsilon^2), \xi = \frac{2}{3})$ *locally list decodable by constant-depth oracle circuits of size* $\mathrm{poly}(r, 1/\varepsilon)$ *that use majority gates of fan-in* $O(1/\varepsilon)$ *and make non-adaptive queries.*

*By a standard amplification argument, using a construction of approximate majority in* $\mathbf{AC}^0$ *(see [ABO84]), the code* Had *is*

$$\left(\rho, Q = O\left(\frac{r}{\varepsilon^2}\right), L\right)$$

*locally list decodable by constant-depth circuits of size* $\mathrm{poly}(r, 1/\varepsilon)$ *that use majority gates of fan-in* $O(1/\varepsilon)$ *and make non-adaptive queries.*

## 4.2 Proof of Theorem 4.2

We now present the proof of Theorem 4.2. Recall that we are given $k$, $\varepsilon$, and $\gamma$. Set $\delta = \frac{1}{25}, \varepsilon_{\mathsf{Had}} = \frac{\varepsilon}{2}$, $L_{\mathsf{Had}} = O(1/\varepsilon_{\mathsf{Had}}^2)$ where the constant hidden inside the $O$-notation is universal and sufficiently large, and $\varepsilon_{\mathsf{IW}} = \frac{\varepsilon_{\mathsf{Had}}}{L_{\mathsf{Had}}} = \Theta(\varepsilon^3)$.We use the following three codes.

1. $\mathsf{GGHKR}' \colon \{0,1\}^k \to \{0,1\}^{k'}$ is the code given in Theorem 4.4, with $k' = k^{c_1}$ for some universal constant $c_1 > 1$, such that $\mathsf{GGHKR}'$ is locally uniquely decodable from $1 - \delta$ fraction of agreement by $\mathbf{TC}^0$ circuits of size $s_{\mathsf{GGM}} = 2^{\mathrm{polyloglog}(k)}$. Note that the decoder circuit makes at most $Q_{\mathsf{GGM}} = s_{\mathsf{GGM}}$ queries.

2. $\mathsf{IW} \colon \{0,1\}^{k'} \to (\{0,1\}^r)^{k''}$ is the code given in Theorem 4.5, set with the parameter $\delta_{\mathsf{IW}} = \delta$, $\gamma_{\mathsf{IW}} = \gamma/2c_1$, and $\varepsilon_{\mathsf{IW}}$. Note that we can take $k'' = (k'/\varepsilon_{\mathsf{IW}})^{c_2/\gamma}$ and $r = c_2 \log \frac{1}{\varepsilon}$ for some universal constant $c_2 > 1$. Also, recall that IW is

   $$\left(\varepsilon_{\mathsf{IW}} \to 1 - \delta, Q_{\mathsf{IW}} = \mathrm{poly}\left(\log k, \frac{1}{\varepsilon}\right)\right)$$

   locally approximately list decodable by circuits of size

   $$s_{\mathsf{IW}} = O\left(\frac{(k')^{\gamma_{\mathsf{IW}}}}{\varepsilon_{\mathsf{IW}}{}^2}\right) = O\left(\frac{k^{\gamma/2}}{\varepsilon^6}\right)$$

   that have one majority gate of fan-in at most $Q_{\mathsf{IW}}$.

3. $\mathsf{Had} \colon \{0,1\}^r \to \{0,1\}^{2^r}$ is the Hadamard code. Recall that Had is

   $$\left(\frac{1}{2} + \varepsilon_{\mathsf{Had}}, Q_{\mathsf{Had}} = O\left(\frac{r}{\varepsilon_{\mathsf{Had}}^2}\right) = O\left(\frac{\log \frac{1}{\varepsilon}}{\varepsilon^2}\right), L_{\mathsf{Had}}\right)$$

   locally list decodable by $\mathbf{AC}^0$ circuits of size $s_{\mathsf{Had}} = \mathrm{poly}(r/\varepsilon_{\mathsf{Had}}) = \mathrm{poly}(1/\varepsilon)$ that use majority gates of fan-in $O(1/\varepsilon)$.

Our 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 $y = \mathsf{IW}(\mathsf{GGHKR}'(x)) \in \Sigma^{k''}$ for $\Sigma = \{0,1\}^r$, and encode each symbol of $y$ with $\mathsf{Had}$. That is,

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

Note that the output length of $\mathcal{C}$ is indeed

$$n = k'' \cdot 2^r = \left(\frac{k'}{\varepsilon_{\mathsf{IW}}}\right)^{c_2/\gamma} \cdot \frac{1}{\varepsilon^{c_2}} \leq \left(\frac{k}{\varepsilon}\right)^{c_3/\gamma}$$

for a universal constant $c_3 > 1$.

### 4.2.1 Local list decodability of $\mathcal{C}$

We denote the code concatenation of $\mathsf{IW}$ and $\mathsf{Had}$ by $\mathcal{C}'\colon \{0,1\}^{k'} \to \{0,1\}^n$. We first show:

**Lemma 4.7.** *There is a constant $c_4$ such that the code $\mathcal{C}'$ is*

$$\left(\frac{1}{2} + \varepsilon \to 1 - \delta, Q' = \left(\frac{\log k}{\varepsilon}\right)^{c_4}\right)$$

*locally approximately list decodable by constant-depth oracle circuits of size $s' = k^{\gamma/2} \cdot (1/\varepsilon)^{c_4}$ each having at most $Q'$ majority gates of fan-in at most $Q'$ and makes non-adaptive queries.*

**Proof.** Let $A_1, \ldots, A_{L_{\mathsf{IW}}}$ and $B_1, \ldots, B_{L_{\mathsf{Had}}}$ be the decoding circuits that correspond to both codes. For each $j_1 \in [L_{\mathsf{IW}}]$ and $j_2 = [L_{\mathsf{Had}}]$, we let $C_{j=(j_1,j_2)}$ be the following decoding circuit, that on input $i \in \{0,1\}^{k'}$ and query access to $w \in \{0,1\}^n$, acts as follows.

- $C_j$ applies the decoding procedure $A_{j_1}$ on $i$.
- Whenever $A_{j_1}$ queries an element $i' \in [k'']$ expecting an answer in $\{0,1\}^r$, the circuit $C_j$ invokes $B_{j_2}(\cdot)$ over all its $r$ inputs, giving it oracle access to the corresponding $2^r$ positions in $w$, to get a guess for the $i'$-th symbol.

Note that we apply the decoder of $\mathsf{Had}$ with the same advice $j_2$ for each query of $A_{j_1}$. The size of each $C_j$ is thus bounded by

$$O(s_{\mathsf{IW}} + Q' \cdot r \cdot s_{\mathsf{Had}}) = k^{\gamma/2} \cdot \mathrm{poly}(1/\varepsilon).$$

For correctness, let $y \in \{0,1\}^n$ and $x \in \{0,1\}^{k'}$ be such that $y$ and $\mathcal{C}'(x)$ agree on at least $1/2 + \varepsilon$ fraction of their coordinates. Denote $z = \mathsf{IW}(x) \in \Sigma^{k''}$. By an averaging argument, there exists a set $I \subseteq [k'']$ of density at least $\varepsilon/2$ such that for each $i \in I$, $\mathsf{Had}(z_i)$ has at least $1/2 + \varepsilon_{\mathsf{Had}}$ agreement with the corresponding position in $y$.

By the list decoding property of the Hadamard code, for every $i \in I$ there exists $j_2(i) \in [L_{\mathsf{Had}}]$ such that $B_{j_2(i)}$ returns the correct symbol. Thus, by another averaging argument, there exists $j_2^\star \in [L_{\mathsf{Had}}]$ for which at least $\frac{\varepsilon}{2} \cdot \frac{1}{L_{\mathsf{Had}}} = \varepsilon_{\mathsf{IW}}$ fraction of the symbols of $z$ are such that the $B_{j_2^\star}(\cdot)$ agrees with the corresponding symbols in $y$. Since there exists $j_1^\star \in [L_{\mathsf{IW}}]$ such that $A_{j_1^\star}$ can $\delta$-approximately decode from agreement $\varepsilon_{\mathsf{IW}}$, we get that $C_{j=(j_1^\star, j_2^\star)}$ recovers $x'$ that has $1 - \delta$ agreement with $x$. ■

We now describe the decoders for $\mathcal{C} = \mathcal{C}' \circ \mathsf{GGHKR}'$. Each decoding circuit $D_j$ is parameterized by $j \in [L']$ and works as follows. Given $i \in [k]$, and oracle access to a string $y \in \{0,1\}^n$,

- We run the local unique decoder of $\mathsf{GGHKR}'$ on input $i$.
- Whenever the local unique decoder wishes to query some index $z \in [k']$, we run the approximate local list decoder of $\mathcal{C}'$, namely $C_j(z)$, having query access to $y$.

Assume that $x \in \{0,1\}^k$ is such that $y$ agrees with $\mathcal{C}(x)$ in at least $1/2 + \varepsilon$ fraction of coordinates. We are guaranteed that for some $j^\star \in [L']$, $C_{j^\star}$ decodes correctly at least $1 - \delta$ fraction of the symbols of $\mathsf{GGHKR}'(x)$. Thus, when the local unique decoder is given the word $(C_{j^\star}(1), \ldots, C_{j^\star}(k'))$ as its noisy codeword, it essentially queries a word with $1 - \delta$ agreement with $\mathsf{GGHKR}'(x)$ and so it correctly decodes every bit in $x$.

For the complexity of each $D_j$, note that its size is bounded by

$$O\left(s_{\mathsf{GGM}} + Q_{\mathsf{GGM}} \cdot s'\right) = k^\gamma \cdot \mathrm{poly}(1/\varepsilon).$$

Finally, note that the total number of queries made is $Q_{\mathsf{GGM}} \cdot Q' = \mathrm{poly}(\log k, 1/\varepsilon)$, and that since all the decoders are non-adaptive, their composition is also non-adaptive.

### 4.2.2 The complexity of $\mathcal{C}$

Finally, we establish the uniform complexity of $\mathcal{C}$.

**Claim 4.8.** *The code $\mathcal{C}$ is computable in logspace. Namely, for any $x \in \{0,1\}^k$ and agreement parameter $\varepsilon > 0$, $\mathcal{C}(x)$ is computable in space $O(\log n) = O(\log(k/\varepsilon))$.*

**Proof.** Recall that $\mathcal{C}$ is the concatenation of $\mathsf{IW} \circ \mathsf{GGHKR}'$ with the Hadamard code. By Theorem 4.4 (proven in Appendix B.1), $\mathsf{GGHKR}'$ is computable in space $O(\log k') = O(\log k)$. By Theorem 4.5, $\mathsf{IW}$ is computable in space $O(\log(rk'')) = O(\log(k/\varepsilon))$. By composition of space-bounded algorithms (Proposition 3.2), $\mathsf{IW} \circ \mathsf{GGHKR}'$ is computable in space $O(\log(k/\varepsilon))$. Finally, note that each coordinate of $\mathcal{C}(x)$ is simply a bit in the encoding of $\mathsf{Had}(y_j)$ for some $j \in [k'']$, where $y = (\mathsf{IW} \circ \mathsf{GGHKR}')(x)$. So overall (again using composition of space-bounded algorithms), $\mathcal{C}$ is logspace computable. ■

# 5 Black-Box Derandomization with Minimal Memory Footprint

In this section we give our black-box derandomization result we outlined in Sections 1.2 and 2.2. In Section 5.2 we prove that $\mathbf{BPL} = \mathbf{L}$ (or alternatively, that $\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[O(S)]$) follows from hardness against *shallow* circuits (as described in Sections 1.1 and 2.1). Section 5.3 is devoted to the efficient derandomization result, which will start with a discussion on our cryptographic hardness assumptions.

## 5.1 A logspace generator with $\mathbf{TC}^0$ reconstruction

We give the standard Nisan–Wigderson generator [NW94], in the standard terminology of reconstructive PRGs. We put emphasis on both the uniform complexity of the generator and on the nonuniform complexity of the reconstruction circuit.

**Theorem 5.1** (Nisan–Wigderson PRG with $\mathbf{TC}^0$ 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 exist two algorithms $G, R$, that for any $f \in \{0,1\}^N$ and for $M = N^{\varepsilon_{\mathsf{NW}}}$, satisfy:*

1. **(Generator.)** *When given input $s \in \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\cdot \log N}$ and oracle access to $f$, the generator outputs an $M$-bit string $G^f(s)$. On input $s$ and $i \in [M]$, $G^f(s)_i$ can be computed in space $(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log N$.*
2. **(Reconstruction.)** *For any $\frac{1}{M}$-distinguisher $D\colon \{0,1\}^M \to \{0,1\}$ for $G^f$ there exists a constant-depth oracle circuit $R$ of size $M^{c_{\mathsf{NW}}}$ that has majority gates, makes non-adaptive queries, and satisfies the following: When given input $x \in [N]$ and oracle access to $D$, the circuit $R$ outputs $f_x$.*

**Proof.** Let $\bar{f}$ be the encoding of $f$ by the code from Theorem 4.2, set with parameters $\varepsilon = 1/O(M^2)$ and $\gamma = \varepsilon_{\mathsf{NW}}$, and note that $\bar{f}$ is of length $\bar{N} = N^{c/\varepsilon_{\mathsf{NW}}}$ for some universal constant $c > 1$. Without loss of generality, we can assume that $\bar{N}$ is a power of two (by padding $\bar{N}$ appropriately).

Let $S_1, \ldots, S_k \subseteq [d]$ be the logspace-computable $(\ell = \log \bar{N}, \alpha\ell)$-design for $\alpha = (\bar{c}/c)\varepsilon_{\mathsf{NW}}{}^2$, guaranteed to us by Theorem 3.11, where $\bar{c}$ is the universal constant given in that theorem. Note that

$$k = 2^{(\alpha/\bar{c})\cdot\ell} = \bar{N}^{\alpha/\bar{c}} = M,$$

and

$$d = \frac{\bar{c}}{\alpha} \cdot \ell = \frac{c^2}{\varepsilon_{\mathsf{NW}}{}^3} \cdot \log N.$$

The generator $G$ gets a seed $s \in \{0,1\}^d$, and for each output index $i \in [M]$ is defined by

$$G^f(s)_i = \bar{f}_{s\restriction_{S_i}},$$

where $s\restriction_{S_i} = \mathsf{Des}(s,i)$ is the projection of $s$ to the $\ell$ coordinates specified by $S_i \subseteq [d]$.

**Complexity of computing each output bit.** To compute the function $(s,i) \mapsto G^f(s)_i$, the oracle machine (which has access to $f$) acts as follows. First, it computes the string $q = \mathsf{Des}(s,i) \in \{0,1\}^\ell$ to the work tape.[18] Then, computing $\bar{f}_q = \mathcal{C}(f)_q$ is done as follows. We maintain, on a dedicated worktape, the location of the required bit to be read from $f$.

[18] Note that we could have also used space-efficient composition and not store the entire $q$ on the work tape, but we won't need this saving.

Whenever the (logspace) encoding function queries the input, we update that worktape to some $k \in [N]$ and query the oracle for $f_k$.

The foregoing procedure can be implemented in space complexity $O((\log N)/\varepsilon_{\mathsf{NW}})$. To see this, note that Des is computable in space $O(\ell) = O((\log N)/\varepsilon_{\mathsf{NW}})$ and we keep those $\ell$ bits on the worktape. The encoding function $\mathcal{C}$ is computable in space $O(\log \bar{N}) = O((\log N)/\varepsilon_{\mathsf{NW}})$, and maintaining the oracle queries, using composition of space-bounded algorithms, takes another $O(\log \ell + \log N)$ space.

**Reconstruction.** Let $D$ be a $\frac{1}{M}$-distinguisher for $G^f(s)$. By a standard analysis following [NW94], there exists an algorithm $R_{\mathsf{NW}}$ that gets advice $\mathtt{adv}$ (that we will specify below) and satisfies

$$\Pr_{x \in \{0,1\}^\ell}\left[R_{\mathsf{NW}}^{D,\mathtt{adv}}(x) = \bar{f}_x\right] \geq \frac{1}{2} + \varepsilon,$$

and $R_{\mathsf{NW}}$ can be implemented by an $\mathbf{AC}^0$ circuit of size $\mathrm{poly}(\ell)$. The advice $\mathtt{adv}$ consists of the combinatorial design, an index $i \in [M]$, values $z \in \{0,1\}^{d-\ell}$, a bit $\sigma \in \{0,1\}$, and the (at most) $M-1$ partial truth-tables of $\bar{f}$ defined by $(i,z)$, where each truth-table is of length at most $2^{\alpha \cdot \ell}$.

Let $\tilde{f}\colon \{0,1\}^\ell \to \{0,1\}$ be the function computed by $R_{\mathsf{NW}}^{D,\mathtt{adv}}(\cdot)$. By Theorem 4.2, there exists a constant-depth circuit of size $N^\gamma \cdot (1/\varepsilon)^c = M^{3c}$ with majority gates that computes $f$ when given non-adaptive oracle access to $\tilde{f}$.Plugging in $R_{\mathsf{NW}}$ and $\mathtt{adv}$ into this circuit, we obtain an oracle circuit (that makes queries only to $D$) of size at most

$$\underbrace{M^{3c}}_{\text{threshold circuit}} + \underbrace{\mathrm{poly}(\ell)}_{R_{\mathsf{NW}}} + \underbrace{M \cdot \ell \cdot \log d}_{\text{design}} + \underbrace{M \cdot 2^{\alpha \cdot \ell}}_{\text{partial truth-tables of } \bar{f}} + \underbrace{d + \log M}_{\text{values } z\text{, index } i}$$

which is at most $M^{c'}$ for a sufficiently large universal constant $c' > 1$. ∎

## 5.2 Warm-up: BPL = L from weaker assumptions

We are now ready to establish our hardness-to-pseudorandomness result, which follows the standard Nisan–Wigderson analysis. We will mostly emphasize where our new hardness assumptions come into play, and only sketch the rest of the argument.

**Theorem 5.2.** *Assume there exists a language $L \in \mathbf{DSPACE}[O(n)]$ that is hard for $\mathbf{TC}^0_{\mathsf{robp}}$ circuits of size $2^{\gamma n}$, where $\gamma \in (0,1)$ is some constant. Then, $\mathbf{BPL} = \mathbf{L}$.*

**Proof.** Let $A \in \mathbf{BPL}$ and fix any $x \in \{0,1\}^n$ for a large enough $n \in \mathbb{N}$. Then, there exists a constant $a > 0$ and an ROBP $C_x\colon \{0,1\}^{M=n^a} \to \{0,1\}$ of width $M$ such that $\Pr_{z \in \{0,1\}^M}[C_x(z) = 1] \geq 2/3$ if $x \in A$, and at most $1/3$ otherwise. Set $\varepsilon_{\mathsf{NW}} = \gamma/c_{\mathsf{NW}}$ where $c_{\mathsf{NW}}$ is the constant from Theorem 5.1, let $N = M^{1/\varepsilon_{\mathsf{NW}}}$, and let $f\colon \{0,1\}^{\log N} \to$

$\{0,1\}$ compute $L$ on inputs of length $\log N$. By our assumption, $f$ is computable in space $O(\log N) = O(\log n)$. Let

$$G^f \colon \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \log N = O(\log n)} \to \{0,1\}^M$$

be the NW PRG given in Theorem 5.1, and assume towards a contradiction that $C_x$ is an $\frac{1}{M}$-distinguisher for $G^f$. Then, there exists a constant-depth threshold oracle circuit $R^{C_x}$ of size $M^{c_{\mathsf{NW}}} = N^{\gamma}$ that computes $f$. Observing that $R^{C_x} \in \mathbf{TC}^0_{\mathsf{robp}}$, this contradicts the hardness of $f$. Hence, $G^f$ fools $C_x$ and we can estimate $\Pr_{z \in \{0,1\}^M}[C_x(z) = 1]$ in space $O(\log n)$, using the fact that $G^f(s)$ is computable in logspace and the seed of the PRG is logarithmic. ■

## 5.3 Minimal-footprint derandomization results

### 5.3.1 A logspace-computable cryptographic PRG with polynomial stretch

For our efficient derandomization result, we will use a cryptographic PRG computable in logarithmic space.[19]

**Assumption 5.3** (a PRG computable in space sub-logarithmic in the output length)**.** *There exists a constant $c_{\mathsf{cry}} > 1$ such that for every $\eta > 0$ and $C \in \mathbb{N}$ there exists an algorithm $G^{\mathsf{cry}}$ that gets input $\bar{s} \in \{0,1\}^{n^\eta}$ and outputs $G^{\mathsf{cry}}(\bar{s}) \in \{0,1\}^{n^C}$ such that:*

1. **(Efficiency.)** *The mapping of $(\bar{s}, i) \in \{0,1\}^{n^\eta} \times [n^C]$ to $G^{\mathsf{cry}}(\bar{s})_i$ is computable in space $c_{\mathsf{cry}} \cdot \log |\bar{s}| + o(\log n)$.*
2. **(Pseudorandomness.)** *For every circuit $D \colon \{0,1\}^{n^C} \to \{0,1\}$ of size polynomial in $n$ it holds that $\Pr_{s \in \{0,1\}^{n^\eta}}[D(G^{\mathsf{cry}}(\bar{s})) = 1] \in \Pr_{x \in \{0,1\}^{n^C}}[D(x) = 1] \pm 1/N$.*

**A candidate PRG.** A candidate construction of such a PRG follows from Goldreich's candidate of one-way functions based on expander graphs [Gol11d], with an appropriate choice of expander and of predicate. The candidate PRG $G^{\mathsf{exp}} \colon \{0,1\}^{n^\eta} \to \{0,1\}^{n^C}$ is constructed as follows. Let $\Gamma \colon [n^C] \times [d] \to [n^\eta]$ be the neighbor function of a *lossless expander* with the following guarantee: For any set $S \subseteq [n^C]$ of size at most $|S| \leq n^{(1-\alpha)C}$, it holds that $|\Gamma(S)| \geq (1-\alpha)d|S|$, where $\alpha > 0$ is a small enough constant. Let $P \colon \{0,1\}^d \to \{0,1\}$ be a predicate. Then, given $s \in \{0,1\}^{n^\eta}$ and $i \in [n^C]$,

$$G^{\mathsf{exp}}(s)_i = P\left(s{\restriction}_{\Gamma(i)}\right),$$

where $s{\restriction}_{\Gamma(i)}$ is the restriction of $s$ to the set of right-neighbors of $i$ in $\Gamma$.

An extensive line of research discovered conditions that are necessary for Goldreich's construction to be pseudorandom (see [Ale11; MST06; ABW10; BQ12; ABR16; BR13;

[19] Recall that when we say a function $f \colon \{0,1\}^n \to \{0,1\}^m \equiv \{0,1\}^n \times [m] \to \{0,1\}$ is computable in logspace, we mean that it is computable in deterministic space $O(\log n + \log \log m)$.

CEM+14; OW14; FPV15; AL18; AR16]). For example, the stretch $n^C$ must satisfy (roughly) $C \leq d/3$ (see [MST06; OW14]); the predicate $P$ needs to have degree $\Omega(\ell)$ as a polynomial, and all of its Fourier coefficients of degree $\leq d$ must be zero (see [AL18]); and either the predicate $P$ or the neighbor functions $\Gamma_i$ (for $i \in [d]$) must have sufficiently high circuit complexity (see [OST22]).

For our derandomization result, we need an *explicit*, space-efficient $\Gamma$, and we take $\Gamma$ to be the recent construction by Kalev and Ta-Shma [KT22]. The parameters of their construction matches the well-known GUV lossless expander [GUV09], but we choose to use [KT22] since its logspace computability follows quite easily. We also note that weaker space requirement from $\Gamma$ would also suffice, namely any sub-linear space complexity.

We state the expander's parameters already with respect to the designated parameters of the PRG.

**Theorem 5.4** ([GUV09; KT22]). *For any constants $0 < \eta < C$ and $\alpha \in (0,1)$, and for every $n \in \mathbb{N}$, there exists an expander $\Gamma\colon [n^C] \times [d] \to [n^\eta]$ with $d = \mathrm{polylog}(n)$, such that for any set $S \subseteq [n^C]$ of size at most $|S| \leq n^{(1-\alpha)C}$, it holds that $|\Gamma(S)| \geq (1-\alpha)d|S|$. Furthermore, the function $\Gamma$ is computable in logarithmic space (namely in $O(\log\log n)$ space).*

We will analyze the space complexity of $\Gamma$ in Appendix B.3. Thus, given Theorem 5.4, computing $G^{\mathsf{exp}}(\bar{s})_i$ takes $O(\log d + \log\log n + s_P)$ space (following Proposition 3.2), where $s_P$ is the space required to compute $P$ on inputs of length $d = \mathrm{polylog}(n)$. Hence, for any suitable $P$ that is computable in a small enough sub-linear space, the efficiently require-ment of Assumption 5.3 holds.

In particular, we can take the following candidate of $P$ from [AR16]

$$P(x_1,...,x_d) = (\oplus_{i\in[\lfloor d/2\rfloor]} x_i) \oplus (\mathsf{MAJ}\{x_i : i \in \{\lfloor d/2\rfloor + 1,...,d\}\}) ,$$

which has high enough degree as a rational function, high enough Fourier degree, and large enough circuit complexity (see [AR16; OST22]).

### 5.3.2 Derandomization that doubles the memory footprint

We are now ready to state and prove our derandomization result that doubles the memory footprint, from non-uniform hardness assumptions, whose informal version was given in Theorem 2.

**Theorem 5.5** (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. Suppose that Assumption 5.3 is true, and 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 constant-depth threshold 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 $L$. For any $n \in \mathbb{N}$, let $N = n^2$, and let $f \in \{0,1\}^N$ be the truth-table of $L^{\mathsf{hard}}$ on inputs of length $\ell = \log N$. Let

$$\mathsf{NW}^f : \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\ell} \to \{0,1\}^{N^{\varepsilon_{\mathsf{NW}}}}$$

be the generator from Theorem 5.1, instantiated with $\varepsilon_{\mathsf{NW}} = \frac{\varepsilon}{2c_{\mathsf{NW}} \cdot c_{\mathsf{cry}}}$. Let

$$G^{\mathsf{cry}} : \{0,1\}^{N^{\varepsilon_{\mathsf{NW}}}} \to \{0,1\}^{n^C}$$

be the generator from Assumption 5.3, instantiated with the parameters $\eta = 2\varepsilon_{\mathsf{NW}}$ and $C$.

Denoting the number of seeds of $\mathsf{NW}^f$ by $\bar{N} = N^{c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}} = n^{4c_{\mathsf{NW}}^2 \cdot c_{\mathsf{cry}}/\varepsilon}$, we define $G^f : \{0,1\}^{\log \bar{N}} \to \{0,1\}$ by

$$G(s) = G^{\mathsf{cry}}\left(\mathsf{NW}^f(s)\right).$$

We define the following oracle machine $\bar{M}$: Given input $x \in \{0,1\}^n$ and oracle access to $r' \in \{0,1\}^{n^C}$,

- The machine $\bar{M}$ simulates $M$ on input $x$.
- Whenever $M$ enters a state that flips a random coin, the machine $\bar{M}$ queries the oracle $r'$ in a location corresponding to the current contents of its worktape; that is, $\bar{M}$ writes its current configuration to the oracle tape, and uses it to query a bit in $r' \in \{0,1\}^{n^C}$.[20]

By Claim 3.3, for any $x \in \{0,1\}^n$ it holds that $\Pr_r[M(x,r) = 1] = \Pr_{r'}[\bar{M}^{r'}(x) = 1]$, where $r, r' \sim \mathbf{u}_{n^C}$, and that $\bar{M}$ uses $S + O(\log S)$ space.

Our deterministic algorithm $A = A_L$ for $L$ gets input $x$, and outputs

$$\mathsf{MAJ}_{s \in \{0,1\}^{\log \bar{N}}}\left\{\bar{M}^{G(s)}(x)\right\}. \tag{5.1}$$

**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 $G(s)$. The oracle is implemented by space-bounded composition of $G^{\mathsf{cry}}$, of $\mathsf{NW}^f$, and of the machine for $L^{\mathsf{hard}}$; that is, whenever $\bar{M}$ queries the oracle at location $i \in [n^C]$, the algorithm $A$:

[20] The number of configurations of $\bar{M}$ is slightly larger than $n^C$, namely $n^C \cdot \mathrm{polylog}(n)$. Naturally, this can be remedied by slightly increasing the output length of $G^{\mathsf{cry}}$, even by taking $C+1$ instead of $C$ whenever $n$ is large enough. This does not change the theorem's statement, so for the sake of readability we assume $n^C$ configurations.

1. Simulates the algorithm $M_{\mathsf{cry}}$ that computes $(\bar{s}, i) \mapsto G^{\mathsf{cry}}(\bar{s})_i$, giving it virtual access to $\bar{s} = \mathsf{NW}^f(s) \in \{0,1\}^{N^{\varepsilon_{\mathsf{NW}}}}$ and direct access to $i$ (i.e., using the content of the relevant worktape of $M$).

2. Whenever $M_{\mathsf{cry}}$ queries $\bar{s}$ at location $j \in [N^{\varepsilon_{\mathsf{NW}}}]$, the algorithm $A$ simulates the algorithm $M_{\mathsf{NW}}$ that computes $\mathsf{NW}^f(s)_j$, giving it direct access to $j$. Finally,

3. Whenever $M_{\mathsf{NW}}$ queries $f$ at location $q \in [N]$, the algorithm $A$ runs the machine for $L^{\mathsf{hard}}$, giving it direct access to $q$.

The space complexity of $A$ follows from Proposition 3.2, but for concreteness we spell out the details. Denoting the space complexity of $f$ by $S' = (C+1+\varepsilon+\delta)/2 \cdot \log N = (C+1+\varepsilon+\delta) \cdot \log n$, we have

$$\begin{aligned}
&\underbrace{\log \bar{N}}_{\text{enumerating } s} + \underbrace{S + O(\log S)}_{\bar{M}} + \underbrace{c_{\mathsf{cry}} \cdot \log(N^{\varepsilon_{\mathsf{NW}}}) + o(\log n)}_{M_{\mathsf{cry}}} \\
&\quad + \underbrace{\frac{c_{\mathsf{NW}}}{\varepsilon_{\mathsf{NW}}} \cdot \log N}_{M_{\mathsf{NW}}} + \underbrace{S'}_{\text{computing } f} + \underbrace{\log \bar{N}}_{\text{counting the outcomes of } M} \\
&\quad + \underbrace{c_0 \cdot (\log N + \log(N^{\varepsilon_{\mathsf{NW}}}))}_{\text{composition overhead}} \\
&\quad = S + S' + \frac{c_1}{\varepsilon} \cdot \log n = \left(2C + \frac{c}{\varepsilon} + \delta\right) \cdot \log n,
\end{aligned}$$

where $c_0, c_1, c > 1$ are sufficiently large universal constants.

**Correctness of $A$.** Fix any $x \in \{0,1\}^n$, and let $D_0 = D_{0,x}$ be defined by $D_0(r) = \bar{M}^r(x)$. Recall that

$$\Pr_r[D_0(r) = 1] = \Pr_r[M(x,r) = 1].$$

Since $G^{\mathsf{cry}}$ is $(1/n^C)$-pseudorandom for all circuits of size $\mathrm{poly}(n)$, and in particular for $D_0$, we have that

$$\Pr_r[D_0(r) = 1] \in \Pr_{\bar{s}}\left[D_0(G^{\mathsf{cry}}(\bar{s})) = 1\right] \pm \frac{1}{n^C}.$$

Now, let $D_1 \colon \{0,1\}^{N^{\varepsilon_{\mathsf{NW}}}} \to \{0,1\}$ be defined by $D_1(\bar{s}) = D_0(G^{\mathsf{cry}}(\bar{s}))$. Assume towards a contradiction that

$$\Pr_{\bar{s}}\left[D_1(\bar{s}) = 1\right] \notin \Pr_s\left[D_1\left(\mathsf{NW}^f(s)\right) = 1\right] \pm \frac{1}{10}.$$

By Theorem 5.1, there exists a constant-depth threshold circuit $R$ of size $N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} < N^{\varepsilon}$ that, when given oracle to $D_1$, computes the mapping of $x \in [N]$ to $f_x$, so overall, $A$ decides $L$.

**Claim 5.6.** *$D_1$ can be computed by a machine running in space $\frac{C+1+\varepsilon}{2} \cdot \log N$ with $n$ bits of advice.*

*Proof.* The $n$ bits of advice will simply be the value of $x$. We begin the proof by noting that using standard space-efficient composition of $D_0$ and $G^{\mathsf{cry}}$ will be too wasteful, since it would take

$$\underbrace{S + O(\log S)}_{\bar{M}} + \underbrace{c_{\mathsf{cry}} \cdot \log(N^{\varepsilon_{\mathsf{NW}}}) + o(\log n)}_{M_{\mathsf{cry}}} + \log n + \underbrace{c_0 \cdot \log(n^C)}_{\text{composition overhead}}$$

space, where:

- The second-to-last term, $\log n$, follows from the need to maintain access to $x$. Recall that $x$ is an advice string, and the space complexity bound on the simulation of $\bar{M}$ assumes that $x$ is given on the input tape. We will not try to reuse any of the other cells in the computation, and simply allocate $\log n$ dedicated bits.[21]
- The last term $c_0 \cdot S$, which is too much for us, follows from the fact that we seemingly need to store the coordinate that $D_0$ wishes to read from $G^{\mathsf{cry}}(\bar{s})$.

To address the second bullet point, recall that we *know* the coordinates are read according to the configuration of $\bar{M}$ (i.e., $M$ on $x$). Thus, whenever $D_1$, while simulating $D_0(\cdot)$, needs access to a coordinate $i \in [n^C]$ of $G^{\mathsf{cry}}(\bar{s})$, it will run the mapping $(\bar{s}, i) \mapsto G^{\mathsf{cry}}(\bar{s})_i$. To provide the foregoing mapping with access to $(\bar{s}, i)$, recall that $\bar{s}$ is written on the worktape; and whenever the mapping needs to access a bit in $i$ (i.e., in the string $i \in \{0,1\}^{\log(n^C)}$), $\bar{M}$ will provide it. Specifically, recall that $i$ is the current configuration of $\bar{M}$. We simulate the mapping using a virtual input head, while storing its location using $\log(|\bar{s}| + |i|) < \log(n)$ bits; and whenever the mapping queries a bit $j \in \log(|i|)$ of $i$, we answer with the $j$-th bit in the configuration of $\bar{M}$ (by moving the worktape head to the corresponding location and answering appropriately).

The subtle point is that the stated space complexity of computing $G^{\mathsf{cry}}(\bar{s})_i$ assumes that we have a bi-directional access to $i$. In Section 3.1, we explained that printing the current $i$ can be done with essentially no loss in parameters, but printing $i$ entirely is too costly for us! Instead, to mimic such a bi-directional access, $D_1$ will be able to query *a specific index* of the current configuration of $M$. Luckily, the configuration does not change until until the query to $G^{\mathsf{cry}}(s)_i$ is answered and the execution of $\bar{M}$ continues, so consistency is guaranteed. Overall, such an implementation, in which $D_1$ is also responsible for supplying the bits of each configuration on demand, can be done in space

$$\begin{aligned} &S + O(\log S) + c_{\mathsf{cry}} \cdot \log(N^{\varepsilon_{\mathsf{NW}}}) + o(\log n) + \log n + O(\log\log(n^C)) \\ &< \frac{C}{2} \log N + (1+\varepsilon) \log n, \end{aligned}$$

where the negligible $O(\log\log(n^C))$ term, together with the $O(\log S)$ term, is the space we allocate for the configurations mechanism. □

[21] One can also consider a model in which these $\log n$ bits are not a part of the overall space, but it won't significantly matter to us.

To conclude, note that $D_1$ uses less than $\frac{C+1+\varepsilon}{2} \log N$ space and $n$ bits of non-uniformity and computes $L^{\mathsf{hard}}$, thus contradicting its hardness. The non adaptivity follow directly from Theorem 5.1. ■

By a standard padding argument, we can conclude:

**Corollary 5.7.** *Under the assumptions and notation of Theorem 5.5, for any $S(n) = \Omega(\log n)$, we have that*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[\left(2 + \frac{c/\varepsilon + \delta}{C}\right) S\right].$$

*In particular, if for example the assumptions of Theorem 5.5 hold for an arbitrarily large constant $C$, $\varepsilon = 1/4$ and $\delta = 4$, we have*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[(2+\tau)S]$$

*where $\tau > 0$ is arbitrarily small.*

To conclude, observe that space-bounded algorithms with advice can *simulate* the $\mathbf{TC}^0$ computation as long as the circuit's size is not too large, so hardness against small $\mathbf{TC}^0$ with *non-adaptive* oracle access to space-bounded algorithms with advice is implied by hardness against space-bounded algorithm with (a slightly longer) advice. Indeed, this is how we stated Assumption 1. Formally:

**Claim 5.8.** *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.*

*Proof.* The space-bounded algorithm will simply simulate the circuit $C$, whose encoding will be given to it as advice. A bit more formally, let $F$ be a TM that gets $2^{n/2} + 2^{\varepsilon n}$ bits of advice, where the latter term is the encoding of $C$. Then, $F$ runs the DFS-style algorithm for evaluating a constant-depth circuit. Starting from the output gate, $F$ recursively evaluate each of its children. For each evaluated gate, it simulates a machine implementing the functionality of the gate, while answering its queries to the children (feeding into the gate) by space-bounded composition. Handling MAJ, AND, OR, and NOT, is straightforward. When the evaluated gate is an oracle gate to the algorithm $A$ that get $2^{n/2}$ bits of non-uniformity and runs in space $\frac{C+1+\varepsilon}{2} \cdot n$, $F$ simulates it using its first $2^{n/2}$ bits of advice and the values of the oracle gate's children.

Implementing the above emulation requires recursion of constant depth. The non-adaptivity means that in each path there will be at most one oracle gate. Thus, we need to allocate space for constantly many layers of AND, OR, and MAJ, and one layer for the evaluation of the oracle. As the maximal fan-in over all gates is $2^{\varepsilon n}$, the computation of AND and OR requires $O(1)$ additional space, and the computation of MAJ requires $\varepsilon n$ additional space. Evaluating the oracle takes $\frac{C+1+\varepsilon}{2} \cdot n$ space, plus an additive factor of $O(\varepsilon n)$ to maintain access to the algorithm's input. For bookkeeping, we need to keep a constant number of gate pointers, each of which takes $\varepsilon n$ space. ■

**Using an alternative hardness assumption.** Let us take a closer look at $D_1$ from the proof above, when applied with our specific candidate construction for $G^{\mathsf{cry}}$ – the expander-based PRG. *Without* the cryptographic PRG, we could model $D_1$ simply as an ROBP. Given $G^{\mathsf{cry}}$ and an input $\bar{s}$, however, we can model the computation $D_1(\bar{s})$ by a *read-many* branching program in the following manner.

Letting $\Sigma = \{0,1\}^d$, in our notion of read-many branching programs (RMBPs) we will allow each layer to query $d$ bits of the BP's input. Formally, in an $[n, T, w]_\Sigma$ (oblivious) RMBP $B$, each of the branching program's $T$ layers is still described by $B_i \colon [w] \times \Sigma \to [w]$, but $B$ is also equipped with an assignment $\mathsf{a}_B \colon [T] \to [n]^d$ indicating which bits of the input should be read at each time step. Then, $B$ computes a function on $n$ bits as follows: Start at $v_0$, and then for $i = 1, \ldots, T$, read the input symbol $\sigma = x\restriction_{\mathsf{a}_B(i)}$ and transition to the state $v_i = B_i(v_{i-1}, \sigma)$.

Using the notation of the proof of Theorem 5.5, we start by the simple observation that $D_0$ can be computed by a $[2^S, 2^S]_{\{0,1\}}$ ROBP, denote it by $B_0$. That is, at each layer, the input the the BP is some $r_{i_j} \in \{0,1\}$ for distinct $i_1, \ldots, i_{n^C}$. For $D_1$, we need to read $G^{\mathsf{cry}}(\bar{s})_{i_1}, \ldots, G^{\mathsf{cry}}(\bar{s})_{i_{n^C}}$. Recalling our candidate construction for $G^{\mathsf{cry}}$, this amounts to querying the input $\bar{s}$ at $d = \mathrm{polylog}(n)$ coordinates at each layer. Thus, we can compute $B_0$ by another BP $B_1$ over the alphabet $\Sigma = \{0,1\}^d$. However, $B_1$ is not read-once. In particular, $B_1$ is an RMBP of length $2^S$ and width $2^S$ that gets inputs of length $N^{\varepsilon_{\mathsf{NW}}}$.

However, without any *uniformity constraints*, the RMBP can simply compute the hard function, since $2^S \geq 2^n$, so we need to be more careful. In particular, we can require that the $\mathbf{TC}^0$ circuit can be generated by *uniform machines with some $a \ll 2^n$ bits of advice*. That is, we separate the space complexity (represented by the width of the BP) from the amount of non-uniformity $a$. Formally:

**Definition 5.9.** *We say that $L \in \mathbf{TC}^0[s]^{\|\mathsf{BP}[w]}/a$ if there exists a TM $M$ that, on input $1^n$ and $a(n)$ bits of advice, runs in space $O(\log s(n))$ and prints a $\mathbf{TC}^0$ oracle circuit $C$ of size $s(n)$ and an RMBP $P$ of width $w(n)$ that satisfies the following: For every $x \in \{0,1\}^n$ it holds that $C^{\|P}(x) = L(x)$ (that is, the calls to $P$ are non adaptive).*

In Section 6.1 we discuss the uniformity of the $\mathbf{TC}^0$ decoder and the number of random bits needed to generate it (see Theorem 6.1). Fixing them as advice, we can refine the hardness assumption from Theorem 5.5 to the following one:

**Assumption 5.10.** *For any two constants $\varepsilon \in (0,1)$ and $C \in \mathbb{N}$, there exists*

$$L^{\mathsf{hard}} \in \mathbf{DSPACE}\left[\frac{C+1+\varepsilon+\delta}{2} \cdot n\right]$$

*for some small constant $\delta > 0$ hard is hard for $\mathbf{TC}^0[2^{\varepsilon n}]^{\|\mathsf{BP}[w]}/a$, where $w = 2^{(C/2)n}$ and $a = 2^{n/2}$.*

We stress that using Assumption 5.10 requires using our candidate Goldreich's PRG in Assumption 5.3.

### 5.3.3 More efficient derandomization from catalytic assumptions

The $2 \cdot S$ term of Theorem 5.5 comes from simulating $\bar{M}$ (which in turn simulates $M$) and computing the hard function $f$. What if those two computations could "share" computation space? A *catalytic* assumption on $f$ would allow us to compute $f$ using the cells used for the computation of $\bar{M}$. Once the specific location of $f$ is computed, the worktape used to simulate $\bar{M}$ is returned to its previous content, and the simulation continues. In order to enact this plan, we will assume the following.

**Assumption 5.11.** *For any two constants $\varepsilon \in (0,1)$ and $C \in \mathbb{N}$ there exists a constant $\delta > 0$ and a language*

$$L^{\mathsf{hard}} \in \mathbf{CSPACE}\left[\frac{\delta}{2} \cdot n, \frac{C+1+\varepsilon+\delta}{2} \cdot n\right]$$

*that is hard for constant-depth threshold circuits of size $2^{\varepsilon n}$ with oracle access to a language in $\mathbf{DSPACE}[\frac{C+1+\varepsilon}{2} \cdot n]/2^{n/2}$.*

**Theorem 5.12** (derandomization with nearly no additional footprint)**.** *Suppose that Assumption 5.3 and Assumption 5.11 are true. Then, for $S(n) = C \cdot \log(n)$, we have that*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[S + \left(\frac{c}{\varepsilon} + 2\delta\right) \log n\right].$$

*Proof sketch.* We follow the same simulation as in Theorem 5.5, but replace the simulation of $f$ with a catalytic simulation of $f$. Towards that end, we will use an additional work tape for the non-catalytic space required to compute $f$.

The correctness readily follows from the proof of Theorem 5.5. For the space complexity, keeping the notation of the proof of Theorem 5.5, we use additional $\frac{\delta}{2} \log N = \delta \log n$ space for the non-catalytic space, and for the simulation of $\bar{M}$ we allocate

$$\max\{S + O(\log S), S'\} = S'$$

cells, that would also suffice for the catalytic computation of $f$. ■

Again, a padding argument shows that under the above assumptions, it holds that

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[\left(1 + \frac{c/\varepsilon + 2\delta}{C}\right) S\right]$$

for all $S(n) \geq C \log n$.

# 6 Non Black-Box Derandomization with Minimal Memory Footprint

## 6.1 A logspace generator with logspace-uniform $\mathrm{TC}^0$ reconstruction

We now prove a more refined version of Theorem 5.1. There, we asserted that any distinguisher for the PRG can be transformed into a relatively small circuit that computes the

hard function. The following refined version asserts that this transformation is *efficient*. Namely, there exists a probabilistic logspace algorithm that prints the circuit. Recall that in terms of parameters, this means that the algorithm runs in space $O(\log N)$ and prints a circuit of size $N^{\Omega(1)}$.

**Theorem 6.1** (NW PRG with logspace-uniform $\mathbf{TC}^0$ 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 exist two algorithms $G, R$, that for any $f \in \{0,1\}^N$ and for $M = N^{\varepsilon_{\mathsf{NW}}}$, satisfy:*

1. **(Generator.)** *When given input $s \in \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\cdot \log N}$ and oracle access to $f$, the generator outputs an $M$-bit string $G^f(s)$. On input $s$ and $i \in [M]$, $G^f(s)_i$ can be computed in space $(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log N$.*
2. **(Reconstruction.)** *Let $D\colon \{0,1\}^M \to \{0,1\}$ be a $\frac{1}{M}$-distinguisher for $G^f$. Then, when given input $1^N$ and oracle access to $f$, the algorithm $R$ runs in space $(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log N$, uses $\widetilde{O}(M^5)$ random coins, and with probability at least $2/3$ 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: When $C$ gets input $x \in [N]$ and oracle access to $D$ it outputs $f_x$.*

To prove Theorem 6.1 we reanalyze both the Nisan-Wigderson PRG and the code from Theorem 4.2. In what comes next, we first state some necessary technical tools, and then prove the theorem.

### 6.1.1 Refined versions of the codes used in Section 4

Recall that, as described in Section 4, the code underlying Theorem 4.2 combines the codes from [GGH+07] and from [IW97] as well as the Hadamard encoding. We will use the following results about those three codes.

**Proposition 6.2** (uniform decoding of [GGH+07])**.** *The code $\mathsf{GGHKR}'$ of Theorem 4.4 is locally decodable by logspace uniform $\mathbf{TC}^0$ circuits. Namely, there is a deterministic TM that runs in space $\mathrm{polyloglog}(k)$ and outputs a randomized $\mathbf{TC}^0$ oracle circuit $D'$ of size $2^{\mathrm{polyloglog}(k)}$ that locally uniquely decodes $\mathsf{GGHKR}'\colon \{0,1\}^k \to \{0,1\}^{k'}$ from $\rho = \frac{24}{25}$ fraction of agreement with probability at least $1 - \frac{1}{k}$ (over the circuit's internal randomness), while making non-adaptive queries.*

**Proposition 6.3** (uniform decoding of [IW97])**.** *There exists a constant $c_{\mathsf{IW}} > 1$ such that for any two constants $\delta_{\mathsf{IW}}, \gamma_{\mathsf{IW}} > 0$, and every $\varepsilon_{\mathsf{IW}} > 0$, the following holds. Let*

$$\mathsf{IW}\colon \{0,1\}^k \to (\{0,1\}^r)^{k'}$$

*be the code from Theorem 4.5. Then, there exists a randomized oracle machine $R_{\mathsf{IW}}$ that, on input $1^k$ and oracle access to $f \in \{0,1\}^k$, runs in space $\frac{c}{\gamma_{\mathsf{IW}}} \log k$, uses $\frac{c}{\varepsilon_{\mathsf{IW}}^2 \gamma_{\mathsf{IW}}} \log k$ bits of randomness, makes at most $k^{\gamma_{\mathsf{IW}}}$ queries to $f$, and prints a constant-depth oracle circuit $C_{\mathsf{IW}}$ of size $k^{\gamma_{\mathsf{IW}}}/\varepsilon_{\mathsf{IW}}^2$ that has one majority gate, makes non-adaptive queries, and such that the following holds: For any $\tilde{f} \in (\{0,1\}^r)^{k'}$ satisfying $\Pr_{z\in[k']}[\tilde{f}_z = \mathsf{IW}(f)_z] \geq \varepsilon_{\mathsf{IW}}$, with probability at least $0.99$ over the randomness of $R_{\mathsf{IW}}$ it holds that $\Pr_{x\in[k]}\left[C_{\mathsf{IW}}^{\tilde{f}}(x) = f(x)\right] \geq 1 - \delta$.*

**Proposition 6.4** (uniform decoding of Hadamard [GL89]). *For any space-computable $k\colon \mathbb{N} \to \mathbb{N}$ such that $k(\ell_0) \leq \ell_0$ and $\varepsilon\colon \mathbb{N} \to (0,1)$, the following holds.*

- **Encoding.** *Let* $\mathsf{Had}$ *be the transformation that maps any function $g\colon \{0,1\}^{\ell_0} \to \{0,1\}^{k(\ell_0)}$ to the function $\mathsf{Had}(g)\colon \{0,1\}^{\ell_0+k(\ell_0)} \to \{0,1\}$ such that $(\mathsf{Had}(g))(x,z) = \oplus_i g(x)_i \cdot z_i$.*
- **Decoding.** *There exists an algorithm $R_{\mathsf{Had}}$ that, on input $1^{\ell_0}$ runs in probabilistic space $O(\log(\ell_0/\varepsilon))$, where $\varepsilon = \varepsilon(\ell_0)$, uses $\mathrm{poly}(\ell_0/\varepsilon)$ random coins, and with probability at least $1 - 2^{-\ell_0}$ prints an oracle circuit $C_{\mathsf{Had}}$ of constant depth, and size $\mathrm{poly}(\ell_0/\varepsilon)$, that has a majority gate, makes non-adaptive queries, and satisfies the following. For every $\widetilde{\mathsf{Had}(g)}$ that agrees with $\mathsf{Had}(g)$ on $1/2+\varepsilon$ fraction of the inputs, there is a set $X \subseteq \{0,1\}^{\ell_0}$ of density $\rho(X) \geq \varepsilon/2$ satisfying: For every $x \in X$ there exists $i(x) \in [O(1/\varepsilon^2)]$ such that*

$$C_{\mathsf{Had}}^{\widetilde{\mathsf{Had}(g)},i(x)}(x) = g(x).$$

The proof of Proposition 6.2 appears in Appendix B.2. The proof of Proposition 6.3 appears in Appendix A. The proof of Proposition 6.4 follows from [GL89] (see, e.g., [AB09, Proof of Theorem 9.12] or [CT21b, Theorem A.7] for an explanation).[22] Given these statements, we are now ready to prove Theorem 6.1.

### 6.1.2 Proof of Theorem 6.1

We follow the proof of Theorem 5.1 and mention the necessary changes. The construction of the generator $G$ is precisely the same, except that we instantiate the code from Theorem 4.2 with parameter values $\varepsilon/2$ instead of $\varepsilon$ (recall that $\varepsilon = 1/O(M^2)$) and $\delta = 1/50$ instead of $\delta = 1/25$. These changes do not meaningfully affect the construction or analysis of $G$, and thus we only need to prove the claim about the uniform reconstruction algorithm $R$.

Unraveling the proofs of Theorem 5.1 and Theorem 4.2, the generator $G$ uses the following mappings. Given a function $f \in \{0,1\}^N$, it first maps it to $f_1 = \mathsf{GGHKR}'(f) \in \{0,1\}^{N'}$, where $N' = N^{c_1}$ for a universal constant $c_1 > 1$. Then, it maps $f_1$ to $f_2 = \mathsf{IW}(f_1) \in (\{0,1\}^r)^{N''}$, where $N'' = (N'/\varepsilon_3)^{c_2/\varepsilon_{\mathsf{NW}}}$ and $r = c_2 \cdot \log(1/\varepsilon)$ where $c_2 > 1$ is also universal. Next, it maps $f_2$ to $f_3 \in \{0,1\}^{\bar{N}}$, where $\bar{N} = (N/\varepsilon)^{c_3/\varepsilon_{\mathsf{NW}}}$, by encoding each of the $N''$ symbols with the Hadamard code. And finally, it uses the generator of [NW94] to map $f_3$ into a set of pseudorandom strings.

In what follows, whenever we say "with high probability", we mean "with probability at least $1-\zeta$" where $\zeta > 0$ is a sufficiently small universal constant.

[22] The only part in the statement of Proposition 6.4 that is not standard is that the algorithm $R_{\mathsf{Had}}$ that constructs $C_{\mathsf{Had}}$ is probabilistic, whereas $C_{\mathsf{Had}}$ is deterministic and succeeds on all inputs in $X$. In standard presentations of [GL89] (e.g., in [AB09]), the procedure is deterministic, and yields a probabilistic circuit $C_{\mathsf{Had}}$ such that $\Pr[C_{\mathsf{Had}}^{\widetilde{\mathsf{Had}(g)},i(x)}(x) = g(x)] \geq 2/3$ for every $x \in X$. To obtain an algorithm $R_{\mathsf{Had}}$ as in the proposition's statement, we consider an algorithm that prints a version of $C_{\mathsf{Had}}$ with error $2^{-2\ell_0}$ instead of $1/3$ (i.e., the circuit performs naive error-reduction, which does not significantly affect its complexity), and then choose fixed random coins and hard-wires them into the error-reduced circuit.

**Step 1: NW reconstruction.** Let $\ell, d, \alpha$ be as in the proof of Theorem 5.1.[23] Recall that a standard analysis of [NW94] implies the following. When choosing at random $i \in [M]$, values $z \in \{0,1\}^{d-\ell}$, and a bit $\sigma \in \{0,1\}$, with probability at least $1/O(M)$ it holds that $\Pr_{x\in\{0,1\}^\ell}\left[C^D_{\mathsf{NW}}(x) = (f_3)_x\right] \geq 1/2+\varepsilon$, where $C_{\mathsf{NW}}$ is a constant-depth circuit of size at most

$$O(M \cdot 2^{\alpha\cdot\ell}) = O(M \cdot \bar{N}^\alpha) = O(M \cdot N/\varepsilon)^{\alpha\cdot c_3/\varepsilon_{\mathsf{NW}}} < M^{c'} \ ,$$

and $c' > 1$ is a sufficiently large universal constant. Moreover, the circuit $C_{\mathsf{NW}}$ can be printed in space $O(\log N)$ given oracle access to $f$ (see below).

For $t = 1, ..., O(M)$, the algorithm $R$ runs the following procedure: Choose $(i, z, \sigma)$ at random and print the corresponding circuit, denoted $C^t_{\mathsf{NW}}$. (All the $C^t_{\mathsf{NW}}$-s are part of the larger overall circuit that $R$ prints.) Now, the algorithm $R$ prints another part of the circuit, which tests the agreement of each $C^t_{\mathsf{NW}}$ with $\bar{f}$, using random sampling of points in $[\bar{N}]$ and the circuit's oracle to $D$ (which allows it to simulate $(C^t_{\mathsf{NW}})^D$), up to accuracy $\varepsilon/2$ and with confidence $1/M^2$. In more detail, $R$ chooses the randomness for this sampling procedure and hard-wires it into the circuit, and uses its query access to $f$ in order to compute the relevant points in $f_3$ and hard-wire them into the circuit. The final part of the circuit that $R$ prints in this step chooses $t$ such that the agreement of $C^t_{\mathsf{NW}}$ with $f_3$ is maximal (breaking ties arbitrarily). In the rest of the proof, we denote the chosen circuit by $C_{\mathsf{NW}}$.

Turning to the analysis of this step, note that with high probability there exists $j$ such that $\Pr[(C^j_{\mathsf{NW}})^D(x) = (f_3)_x] \geq 1/2 + \varepsilon$ (since the success probability of the reconstruction of [NW94] is $1/O(M)$, and $R$ repeats it for $O(M)$ times). Conditioned on that, with high probability over the coins of $R$, all the estimations by random samplings are correct up to accuracy $\varepsilon/2$ (this uses a union-bound over all $j$). In this case, the output circuit $C_{\mathsf{NW}}$ satisfies $\Pr_x\left[C^D_{\mathsf{NW}}(x) = (f_3)_x\right] \geq 1/2 + \varepsilon/2$. We condition on this event.

The complexity of printing each $C^t_{\mathsf{NW}}$. We argue that for each $t$, the algorithm $R$ can print $C^t_{\mathsf{NW}}$ in space $O(\log(N)/\varepsilon_{\mathsf{NW}})$ with oracle access to $f$, as follows. After choosing $(i, z, \sigma) \in \{0,1\}^{O(\log N)}$ and storing them on its work tapes, the algorithm $R$ iterates over choices for $j \in [M]$, and for each $j$ computes the set $S_i \cap S_j$ (using the fact that the mapping f $j \mapsto S_j$ can be done in space $O(\log N)$; see Theorem 3.11). Then, it iterates over all possible choices for $x^{(j)} \in \{0,1\}^{|S_i\cap S_j|}$, in lexicographical order, and for each fixed $(j, x^{(j)})$, it computes the $(\ell = \log \bar{N})$-bit string $y_{j,x^{(j)}}$ that is obtained by placing $x^{(j)}$ in the locations $S_i \cap S_j$ and $\alpha\restriction_{S_j\setminus S_i}$ in the locations $S_j \setminus S_i$.

Now, recall that circuits are described as lists of gates, where the description of each gate contains its index, its type, and the names of at most two gates that feed into to it. For each $(j, x^{(j)})$ and corresponding $y = y_{j,x^{(j)}}$, the algorithm $R$ queries $f_3$ at location $y$, and prints to its output an additional gate in the description of the circuit: The index of this gate is $(j, x^{(j)}) \in [O(M)] \times [M]$, its type is the constant $(f_3)_y$, and it has no gates feeding into it. Note that $R$ does not have direct access to $f_3$, but it can compute $f_3$ using its oracle access to $f$ (because $f_3$ is simply the encoding of $f$ by the code from Theorem 4.2). Since

[23] That is, $\ell = \log \bar{N}$ and $\alpha = (\bar{c}/c) \cdot \varepsilon^2_{\mathsf{NW}}$ for universal constants $c, \bar{c} > 1$, and $d = (c^2/\varepsilon^2_{\mathsf{NW}}) \cdot \log N$.

there are less than $O(M^2)$ choices for $j$ and $x^{(j)}$, and the code from Theorem 4.2 is computable in space $O(\log(N)/\varepsilon_{\mathsf{NW}})$, this entire step can be done in space $O(\log(N)/\varepsilon_{\mathsf{NW}})$.[24]

Next, the algorithm $R$ prints another set of $M \cdot d$ constant gates, which describe the sets $S_1, ..., S_M \subseteq [d]$ in the design (i.e., the description of each $S_i$ is the indicator vector of the $\ell$-bit set $S_i \subseteq [d]$). This can be done in space $O(d) = O(\log(N)/\varepsilon_{\mathsf{NW}})$, relying on Theorem 3.11 and Theorem 3.9. Finally, the algorithm $R$ prints gates that implement the functionality of the standard reconstruction circuit of the Nisan-Wigderson [NW94] generator (given the fixed choices of $(z, i, \sigma)$), where these gates answer the queries that the standard reconstruction makes to $f_3$ by looking the relevant value up in the $j$-th set of $M$ constant gates.

The final complexity of $R$ in this step. Printing each $C^t_{\mathsf{NW}}$ can be done in $O(\log(N)/\varepsilon_{\mathsf{NW}})$ space, and there are $O(M)$ values for $t$, so printing all of these circuits can be done in space $O(\log(N)/\varepsilon_{\mathsf{NW}})$. The space complexity of the last set of gates, which tests the agreement of each $C^t_{\mathsf{NW}}$ with a predetermined sample of values of $f_3$ (that $R$ chooses in advance and hard-wires into the circuit) and chooses the $t$ that maximizes the agreement, can also be done in space $O(\log(N)/\varepsilon_{\mathsf{NW}})$ with oracle access to $f$ (i.e., as above, we compute values of $f_3$ using oracle access to $f$, in space $O(\log(N)/\varepsilon_{\mathsf{NW}})$). The overall number of coins that $R$ uses in this step is $M \cdot (O(\log(N)) + (1/\varepsilon^2) \cdot \mathrm{polylog}(M)) = \widetilde{O}(M^5)$.

**Step 2: Decoding of IW concatenated with Had.** Recall that $L_{\mathsf{Had}} = O(1/\varepsilon^2_{\mathsf{Had}}) = O(M^4)$.[25] For every $j \in [L_{\mathsf{Had}}]$, the algorithm $R$ prints a circuit $C_j$ that acts as follows. The circuit $C_j$ runs the circuit $C_{\mathsf{IW}}$ from Proposition 6.3, and whenever the latter accesses a symbol (i.e., queries a location in $[N'']$ and expects to get back a string in $\{0,1\}^r$), the circuit answers using the circuit $C_{\mathsf{Had}}$ from Proposition 6.4, when oracle queries of $C_{\mathsf{Had}}$ are answered by the circuit $C_{\mathsf{NW}}$ (for the first oracle function) and by the fixed index $j$ (for the second oracle function). In more detail, for every $j \in [L_{\mathsf{Had}}]$ the algorithm runs the algorithms $R_{\mathsf{IW}}$ and $R_{\mathsf{Had}}$ to print such circuits, and points the queries of the oracle gates of $R_{\mathsf{Had}}$ to be answered by $C_{\mathsf{NW}}$.

Then, the algorithm $R$ prints a circuit that, for every $j \in [L_{\mathsf{Had}}]$, estimates the value $\Pr_{x \in [N'']}\left[C_j^{C_{\mathsf{NW}}}(x) = f_2(x)\right]$ up to an additive error of $\delta/2$ and with confidence $1/M^5$. (In the notation $C_j^{C_{\mathsf{NW}}}$ we omit the distinguisher $D$, for simplicity. We remind the reader that $C_{\mathsf{NW}}$ is an oracle circuit and that we analyze it when given access to $D$.) Specifically, $R$ chooses random sample points in $[N'']$ in advance, queries $f_2$ at these points (this can be done by querying $f$ – see below), and hard-wires the result into the circuit, to estimate the agreement of each $C_j$ with $f_2$. The circuit chooses $\bar{j} \in [L_{\mathsf{Had}}]$ such that the agreement of $C_{\bar{j}}$ with $f_2$ on the sample is maximal.

[24] Note that, when computing $f_3$ from $f$, the algorithm $R$ does not have direct access to $f$ as an input, but only oracle access to $f$. Simulating input access to $f$ in this setting requires only $O(\log(|f|)) = O(\log N)$ additional bits of space, to store the location of the head on the simulated input tape.

[25] In the notation of Theorem 4.6 $L_{\mathsf{Had}}$ denotes the list size for local decoding of the Hadamard code, and in the notation of Proposition 6.4 $L_{\mathsf{Had}}$ denotes the number of possible choices of $i(x)$ for every $x \in \{0,1\}^{\ell_0}$. In both cases, $L_{\mathsf{Had}}$ is exactly the same value.

Turning to the analysis, by Proposition 6.4 there is a set $X \subseteq [N'']$ (where $X$ depends on $C_{\mathsf{NW}}$) of density at least $\varepsilon_{\mathsf{Had}}/2$ such that with high probability over the coins of $R_{\mathsf{Had}}$, for every $x \in X$ there is some $j(x) \in [L_{\mathsf{Had}}]$ for which $C_{\mathsf{Had}}^{C_{\mathsf{NW}},j(x)}(x) = f_2(x)$. Conditioned on the latter event, for some $j^\star \in [L_{\mathsf{Had}}]$ there is a set $X' \subseteq [N'']$ of density at least $\varepsilon_{\mathsf{IW}}$ such that for every $x' \in X'$ it holds that $C_{\mathsf{Had}}^{C_{\mathsf{NW}},j^\star}(x) = f_2(x)$. In this case, with high probability over the coins of $R_{\mathsf{IW}}$ it holds that, when $C_{\mathsf{IW}}$ is given access to $C_{\mathsf{Had}}^{C_{\mathsf{NW}},j^\star}$, it computes $f_1$ correctly on at least $1-\delta$ of the inputs in $[k']$. Conditioned on this event, with high probability over the coins of $R$, the circuit finds $\bar{j}$ such that $\Pr_{x\in[N'']}[C_{\bar{j}}^{C_{\mathsf{NW}}}(x) = f_2(x)] \geq 1 - 1/25$. We condition on this event too.

Since $R_{\mathsf{IW}}$ and $R_{\mathsf{Had}}$ run in space $O(\log(N/\varepsilon)) = O(\log N)$, the space complexity of $R$ is also $O(\log N)$. Also, $R$ needs query access to $f_2$ (to run $R_{\mathsf{IW}}$, and when constructing the circuit that estimates the agreement of each $C_j$ with $f_2$), and it can simulate such query access in space $O(\log N)$ using its oracle access to $f$ (since the mapping of $f$ to $f_2$ is computable in space $O(\log N)$, as part of the encoding algorithm of Theorem 4.2 that computes the mapping $f \mapsto f_1 \mapsto f_2 \mapsto f_3$ in space $O(\log N)$[26]). The number of random coins that $R$ uses in this step is $M^4 \cdot \mathrm{polylog}(N)$, and the size of the circuit that $R$ prints is

$$L_{\mathsf{Had}} \cdot \left((N')^{\gamma_{\mathsf{IW}}}/\varepsilon_{\mathsf{IW}}^2 + (N')^{\gamma_{\mathsf{IW}}} \cdot \mathrm{poly}(r/\varepsilon_{\mathsf{Had}})\right) + \mathrm{poly}(L_{\mathsf{Had}}) < M^{c''},$$

where $c'' > 1$ is a sufficiently large universal constant.

**Step 3: Decoding of GGHKR'.** Finally, the algorithm $R$ runs the machine from Proposition 6.2, which prints an oracle $\mathbf{TC}^0$ circuit $D'$, and replaces the oracle calls of $D'$ by queries to the circuit $C_{\mathsf{IW}}$. Moreover, $R$ chooses in advance randomness for $D'$, and hard-wires it into the circuit. The output of $D'$ will be the final output of the overall circuit that $R$ prints. By a union-bound over the $N$ coordinates of $f$, with high probability it holds that the overall circuit computes $f$ correctly on all $N$ coordinates.

The space complexity of $R$ in this step is $O(\log N)$ (the machine from Proposition 6.2 runs in even smaller space), the number of random coins that it uses is $\mathrm{polylog}(N)$, and the last part of the circuit that is printed at this step is just of size $\mathrm{polylog}(N)$. Thus, the overall size of the circuit is bounded by $M^{c'''}$ for a universal constant $c''' > 1$. Note that since each of the components in the circuit makes non-adaptive queries, the overall circuit also makes non-adaptive queries.

## 6.2 Minimal-footprint derandomization results

**Theorem 6.5** (non-black-box derandomization that doubles the memory footprint)**.** *There exist universal constants $c, c'', c_D > 1$ such that for any two constants $\varepsilon \in (0,1)$ and $C \in \mathbb{N}$ the following holds. Let $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be a sequence of distributions, where each $\mathbf{x}_n$ is over $\{0,1\}^n$,*

[26] As in the analysis of Step 1, indeed $R$ does not have input access to $f$ but rather only oracle access, but it can simulate input access using oracle access (i.e., track the location of the input head) with additional $O(\log|f|) = O(\log N)$ bits of memory.

*and let* $\mu\colon \mathbb{N} \to [0,1)$. *Suppose that Assumption 5.3 is true, and that there exist* $\delta > 0$ *and a function* $f\colon \{0,1\}^\star \to \{0,1\}^\star$ *mapping* $n$ *bits to* $N = n^2$ *bits that satisfy the following.*

1. **(Upper bound.)** *The function* $f$ *is computable on inputs of length* $n$ *in deterministic space* $(C + 1 + c\varepsilon + \delta) \cdot \log n$.
2. **(Lower bound.)** *For every probabilistic algorithm* $R$ *running in space* $(C+1+\delta)\cdot\log n + (c''/\varepsilon)\cdot\log n$, *and every sufficiently large* $n \in \mathbb{N}$, *with probability at least* $1-\mu(n)$ *over* $x \sim \mathbf{x}_n$ *the following holds: The probability over the randomness of* $R$ *that it prints a Turing machine of description length* $\sqrt{N}$ *that, when given input* $i \in [N]$ *outputs* $f(x)_i$ *in space* $(C+1+c\varepsilon)\cdot\log n$, *is less than* $2/3$.

*Then, for* $S(n) = C \cdot \log n$, *for every* $L \in \mathbf{BPSPACE}[S]$ *there exists*

$$L' \in \mathbf{DSPACE}\left[2S + \left(\frac{c_D}{\varepsilon} + \delta\right)\cdot\log n\right]$$

*such that for all sufficiently large* $n \in \mathbb{N}$,

$$\Pr_{x\sim\mathbf{x}_n}\left[L'(x) \neq L(x)\right] \leq \mu(n)\ .$$

Note that Theorem 3 is the special case of Theorem 6.5 obtained by using any distribution $\mathbf{x}$ with full support (say, the uniform one) and $\mu \equiv 0$.

**Proof.** Let $L \in \mathbf{BPSPACE}[S(n)]$, and let $M$ be a randomized space-$S$ machine that decides $L$. Given input $x \in \{0,1\}^n$, the derandomization algorithm $A = A_L$ computes $f = f(x) \in \{0,1\}^N$, lets $\mathsf{NW}^f$ and $G^{\mathsf{cry}}$ be exactly as in the proof of Theorem 5.5 (except that now they are defined with respect to $f = f(x)$), and outputs

$$\mathsf{MAJ}_{s\in\{0,1\}^{\log \bar{N}}}\left\{\bar{M}^{G(s)}(x)\right\}\ ,$$

where the definitions of $\bar{M}, \bar{N}, G$ are as in the proof of Theorem 5.5.[27]

**Analysis.** To bound the space complexity of $A$, we follow the same argument as in the proof of Theorem 5.5, with the following modification: Whenever $M_{\mathsf{NW}}$ queries $f$ at location $q \in [N]$, the algorithm $A$ computes $f = f(x)$ and outputs the $q$-th bit. By our bound on the space complexity of $f$ (i.e., $(C + 1 + c\varepsilon + \delta)\cdot\log n$), this change does not affect the overall calculation, and thus $A$ runs in space $(2C + (c_A/\varepsilon) + \delta)\cdot\log n$ for some universal constant $c_A$.

[27] Formally, here we use the generator from Theorem 6.1 whereas in Theorem 5.5 we used the generator from Theorem 5.1, but (as mentioned in the proof of Theorem 6.1) this is precisely the same generator. Indeed, the only difference between the derandomization in Theorem 5.5 and the current derandomization is that in the former, $f$ was the truth-table of a hypothesized hard function, whereas in the latter, $f = f(x)$ is the evaluation of a hypothesized hard function at the specific input $x$.

Now let us prove the correctness of the derandomization. Recall that (by the properties of $G^{\mathsf{cry}}$, and as in the proof of Theorem 5.5), for every $x \in \{0,1\}^n$ it holds that

$$\Pr_r[M(x,r)=1] \in \Pr_{\bar{s}}\left[M(x,G^{\mathsf{cry}}(\bar{s}))=1\right] \pm \frac{1}{n^C} .$$

For every $x \in \{0,1\}^n$, we define $D_x(\bar{s}) = M(x, G^{\mathsf{cry}}(\bar{s}))$. We say that $x \in \{0,1\}^n$ is useful if it satisfies

$$\Pr_{\bar{s}}\left[D_x(\bar{s})=1\right] \notin \Pr_s\left[D_x\left(\mathsf{NW}^f(s)\right)=1\right] \pm \frac{1}{10} .$$

We construct a probabilistic algorithm $R'$ that is an input $x \in \{0,1\}^n$, and – when $x$ is useful – prints, with high probability, a Turing machine $F$ that computes the function $i \mapsto f(x)_i$. Informally and at a high-level, the algorithm $R'$ runs the reconstruction algorithm $R$ from Theorem 6.1, whose output is a circuit $C$, while answering the queries of $C$ to $f = f(x)$ by directly computing $f(x)$ from the input $x$. The Turing machine $F$ has $C$ "hard-coded" into its description. Given input $q \in [N]$, $F$ evaluates $C$ using the standard simulation of $\mathbf{NC}^1 \supseteq \mathbf{TC}^0$ in space $O(\log|C|)$, and whenever $C$ queries the distinguisher, the machine $F$ computes the distinguisher $D$ above (i.e., it simulates $G^{\mathsf{cry}}$ and $M$). The computational bottleneck of $F$ in terms of space complexity is computing $M$, and the main information needed to be encoded in the description of $F$ is $x$ itself (which is of size $n = \sqrt{N}$), and the encoding of $C$ (which is of size $N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} \ll n$). Details follow.

Description of $R'$. Given $x \in \{0,1\}^n$, the algorithm $R'$ simulates the algorithm $R$ from Theorem 6.1, and whenever $R$ queries a location $q \in [N]$, the algorithm $R'$ computes the mapping $x \mapsto f(x)$ and answers with the $q$-th coordinate. Recall that $R$ outputs an oracle circuit $\mathcal{C}$ of size $N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}$; the algorithm $R'$ prints special states of the machine $F$ that encode the gates of $\mathcal{C}$.

We will now specify precisely how the foregoing special states are printed. Our goal is to print them in a way that the transition function of the machine will be easily computable; that is, a priori, the transition function could be any function $[N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}] \to [N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}]$, but we would like this function to have a concise representation that we can easily print. Thus, for every gate $g$ that $R$ prints, $R'$ prints a sequence of $O(\log N)$ states of $F$ that satisfy the following. When $F$ enters into this sequence, it reads the content $\lambda \in [N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} + O(1)]$ of a dedicated worktape, and depending on $\lambda$, writes down (on another dedicated worktape) either the index and type of $g$, or the $\lambda$-th gate that feeds into $g$. The index of each state in the sequence is a string with prefix $g \in [N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}]$ and a suffix in $[N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} + O(\log N)]$, indicating its position in the sequence. Note that the part of the transition function corresponding to these states is indeed easily computable: When the machine enters the sequence corresponding to $g$, the transition reads $\lambda$ bit-by-bit, and with each bit updates the index of the current state according to $\lambda$; after it reaches the state indexed by $(g,\lambda)$ and a trivial suffix $1 \in [O(\log N)]$, it enters a fixed sequence of $O(\log(N))$ steps indexed by $(g,\lambda,1),...,(g,\lambda,O(\log(N)))$ (with a transition function that does not depend on the input) that prints the appropriate string (i.e., either the name of the $\lambda$-th child of $g$, or the type and index of $g$).

Next, the algorithm $R'$ prints another set of states, which, loosely speaking, encode the input $x$. Specifically, it prints $n$ sequences of states, whose indices are in the range $[N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} \cdot O(\log n), N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}} \cdot O(\log n) + O(n)]$, such that for $j \in [n]$, when entering the $j$-th sequence, the machine writes down $x_j$ on a dedicated worktape. This is done using the same approach as above.

Having printed the two special sets of states of $F$ described above, the algorithm $R'$ prints the main functionality of $F$, which is specified by the following uniform algorithm. The machine $F$ runs the DFS-style algorithm for evaluating a constant-depth threshold circuit of the dimensions (i.e., size and depth) of $C$. Starting from the output gate,[28] the machine $F$ recursively evaluates each of its children. For each evaluated gate $g$, it simulates a machine implementing the functionality of the gate, while answering its queries to the children (feeding into the gate) by space-bounded composition; that is:

1. If $g$ is a Majority gate, then the machine $F$ maintains an integer $S$ at this level of recursion, which is initiated to zero. The machine $F$ enumerates over $\lambda \in [N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}]$ (which represents the potential children of the gate) and for each $\lambda$, it enters the sequence of states corresponding to $g$ in order to deduce the index and type of the $\lambda$-th child, then recurses into evaluating this child of $g$. When returning from recursion, it adds the value of the child to $S$. After enumerating over all children, it checks whether $S$ exceeds half the number of children of $g$ (i.e., the last value of $\lambda$ for which a child existed, divided by two) and returns the corresponding answer.

2. If $g$ is an AND gate (resp., an OR gate), then the machine $F$ acts exactly as above, except that in the end it checks whether the value of $S$ is $\lambda$ (resp., is positive).

3. When the evaluated gate $g$ is an oracle call, the machine $F$ simulates the machine $D$ computing $\bar{s} \mapsto M(x, G^{\mathsf{cry}}(\bar{s}))$. Whenever $D$ tries to access a location in $x$, the machine $F$ computes the mapping $j \mapsto x_j$ using the special states described above. Whenever $D$ accesses its input (i.e., the query made to the oracle gate) at location $\lambda \in [N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}]$, the machine $F$ uses space-bounded composition to evaluate the $\lambda$-th child of $g$.

Analysis of $R'$. The space complexity of $R'$ is dominated by the first step, where it simulates $R$ and answers its queries to $f = f(x)$. Since $R$ runs in space $(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log N$, and using space-bounded composition, the space complexity of $R'$ is

$$\frac{c'}{\varepsilon} \cdot \log n + (C + 1 + c\varepsilon + \delta) \cdot \log n \leq \left( C + 1 + \frac{c''}{\varepsilon} + \delta \right) \log n,$$

where $c', c'' > 1$ are universal constants. The complexity of printing the states of $F$ upon reading the gates of $C$, as well as the complexity of "hard-wiring" $x$, is absorbed in the first term by taking a slightly larger $c''$.

[28] We can assume that the output gate was found when running $R$ in the first step of $R'$, and that the first states of $F$ write down its index on a dedicated worktape.

Analysis of $F$. Note that the number of states in $F$ is dominated by the set of states that implement the functionality $j \mapsto x_j$. This is because the number of states required for encoding $\mathcal{C}$ is $\widetilde{O}(|\mathcal{C}|) = \widetilde{O}(N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}) \ll n$, and since the main functionality of $F$ is completely uniform. Thus, the description of $F$ is of size $O(n) = O(\sqrt{N})$.

Let us now analyze the space complexity of $F$. On input $q \in [N]$, implementing the DFS-style algorithm requires recursion of constant depth. Since the circuit is non-adaptive, in each path in the recursion there will be at most one oracle gate. Thus, to implement the recursion, we need to allocate space for constantly many layers of AND, OR, and MAJ, and for one layer where the mapping $\bar{s} \mapsto M(x, G^{\mathsf{cry}}(\bar{s}))$, which we denoted by $D_x(\cdot)$, will be computed. In more details, let $t \leq N^{\varepsilon_{\mathsf{NW}}}$ be the maximal fan-in over all the gates encoded in $F$. Then:

- The computation of AND and OR requires $O(1)$ additional space, and the computation of MAJ requires $\log t$ additional space.
- The computation of $D_x(\bar{s})$, following Theorem 5.5, takes $\frac{C}{2} \log N + \varepsilon \log n$ space. To that, we need to add an additive $\log n + O(1)$ factor to simulate the access to $x$ (which unlike $s$, is not fed to the computation by previous layers).
- For bookkeeping, we need to keep a constant number of gate pointers.

Denoting the depth of the circuit by $c_d$ (which is independent of the constant $C$), the space required to compute $F(j)$ is at most

$$O(\varepsilon_{\mathsf{NW}} \log N) + c_d \cdot \log t + \frac{C}{2} \log N + (1+\varepsilon) \log n < \frac{C + 1 + c'''\varepsilon}{2} \cdot \log N$$

for a universal constant $c''' > 0$, *independent* of the constants $C, c$ so in particular we can assume that $c \geq c'''$. Lastly, note that we need to verify that we embed $\mathcal{C}$ (i.e., the output of $R$) as states in $F$ in a way that allows us to run the simulation while only querying one $D_x$ gate per path. It will be more convenient to require this from $R$ itself, namely that the circuit that $R$ outputs is non adaptive, and indeed this is one of the guaranteed of the reconstruction procedure.

Wrapping up. By the above, if $\Pr_{x \sim \mathbf{x}_n}[x \text{ is useful}] > \mu(n)$, then with probability greater than $\mu(n)$ over choice of $x \sim \mathbf{x}_n$ the following holds: With probability at least $2/3$ over the random coins of $R'$, it prints a Turing machine whose description is of size $O(\sqrt{N})$ and that computes the function $j \mapsto f(x)_j$ in space $(C + 1 + c'''\varepsilon) \log n$. This contradicts the hardness of $f$, and thus we conclude that $\Pr_{x \sim \mathbf{x}_n}[x \text{ is useful}] \leq \mu(n)$. It follows that

$$\Pr_{x \sim \mathbf{x}_n}[A(x) \neq L(x)] \leq \mu(n) \ ,$$

which concludes the proof. ■

Recall that in Assumption 3, the compressed version of $f$ was taken to be of length $\sqrt{|f|}$. Inspecting the above proof, it is clear that we can change $\sqrt{|f|}$ to, say, $|f|^{0.01}$, by increasing the length of $f(x)$ (i.e., the length of the truth table we work with) from $n^2$ to $n^{100}$. For a large enough $C$, this results in a negligible overhead in the derandomization result.

Similar to Section 5.3, by a simple padding argument, we can conclude:

**Corollary 6.6.** *Under the assumptions and notation of Theorem 6.5, for any $S(n) = \Omega(\log n)$, we have that*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}\left[\left(2 + \frac{c_D/\varepsilon + \delta}{C}\right) S\right].$$

*In particular, if for example the assumptions of Theorem 5.5 hold for an arbitrarily large constant $C$, $\varepsilon = 1/4$ and $\delta = 4$, we have*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{DSPACE}[(2+\tau)S]$$

*where $\tau > 0$ is arbitrarily small.*

Assuming, furthermore, that $f$ is computable using mostly catalytic space, along the line of Section 5.3.3, we can reduce the $2S$ factor down to $S$. We omit the details.

# Acknowledgements

We are grateful to Avi Wigderson for several useful conversations regarding the gap between double space blow-up and single space blow-up. We thank Lijie Chen for suggesting the idea of using catalytic space to save on complexity, early in this work, and for pointing out a gap in a previous version of the proof of Theorem 4.2. We also thank Ian Mertz for several useful conversations exploring the abilities of catalytic space.

# 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.

[ABO84] Miklos Ajtai and Michael Ben-Or. "A theorem on probabilistic constant depth computations". In: *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC)*. 1984, pp. 471–474.

[ABR16] Benny Applebaum, Andrej Bogdanov, and Alon Rosen. "A dichotomy for local small-bias generators". In: *Journal of Cryptology* 29.3 (2016), pp. 577–596.

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. "Public-key cryptography from different assumptions". In: *Proc. 42nd Annual ACM Symposium on Theory of Computing (STOC)*. 2010, pp. 171–180.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "Cryptography in $\mathbf{NC}^0$". In: *SIAM Journal on Computing* 36.4 (2006), pp. 845–888.

[AL18] Benny Applebaum and Shachar Lovett. "Algebraic attacks against random local functions and their countermeasures". In: *SIAM Journal of Computing* 47 (1 2018), pp. 52–79.

[Ale11] Michael Alekhnovich. "More on average case vs. approximation complexity". In: *Computational Complexity* 20.4 (2011), pp. 755–786.

[App14] Benny Applebaum. *Cryptography in constant parallel time*. Information Security and Cryptography. 2014, pp. xvi+193.

[AR16] Benny Applebaum and Pavel Raykov. "Fast pseudorandom functions based on expander graphs". In: *Theory of Cryptography. Part I*. Vol. 9985. 2016, pp. 27–56.

[Bar89] David A. Mix Barrington. "Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$". In: *Journal of Computer and System Sciences* 38.1 (1989), pp. 150–164.

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. "Computing with a full memory: catalytic space". In: *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC)*. 2014, pp. 857–866.

[BCP83] Allan Borodin, Stephen Cook, and Nicholas Pippenger. "Parallel computation for well-endowed rings and space-bounded probabilistic machines". In: *Information and Control* 58.1-3 (1983), pp. 113–136.

[BQ12] Andrej Bogdanov and Youming Qiao. "On the security of Goldreich's one-way function". In: *Computational Complexity* 21.1 (2012), pp. 83–127.

[BR13] Andrej Bogdanov and Alon Rosen. "Input locality and hardness amplification". In: *Journal of Cryptology* 26.1 (2013), pp. 144–171.

[CEM+14] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. "On the one-way function candidate proposed by Goldreich". In: *ACM Transactions on Computation Theory* 6.3 (2014), Art. 14, 35.

[Che19] Lijie Chen. "Non-deterministic quasi-polynomial time is average-case hard for **ACC** Circuits". In: *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2019.

[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.

[CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. "Almost-everywhere circuit lower bounds from non-trivial derandomization". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020.

[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.

[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.

[CT22] Lijie Chen and Roei Tell. "When Arthur has neither random coins nor time to spare: Superfast derandomization of proof systems". In: *Electronic Colloquium on Computational Complexity: ECCC* (2022).

[DMO+20] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. "Nearly optimal pseudorandomness From hardness". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020.

[FPV15] Vitaly Feldman, Will Perkins, and Santosh Vempala. "On the complexity of random satisfiability problems with planted solutions". In: *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC)*. 2015, pp. 77–86.

[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.

[GK08] Venkatesan Guruswami and Valentine Kabanets. "Hardness amplification via space-efficient direct products". In: *Computational Complexity* 17.4 (2008), pp. 475–500.

[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 Sample of Samplers: A Computational Perspective on Sampling". In: *Studies in Complexity and Cryptography* 6650 (2011), pp. 302–332.

[Gol11b] Oded Goldreich. "Candidate one-way functions based on expander graphs". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. 2011, pp. 76–87.

[Gol11c] Oded Goldreich. "In a world of $\mathbf{P} = \mathbf{BPP}$". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*. 2011, pp. 191–232.

[Gol11d] Oded Goldreich. "Introduction to Testing Graph Properties". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. 2011, pp. 470–506.

[GSV18] Aryeh Grinberg, Ronen Shaltiel, and Emanuele Viola. "Indistinguishability by adaptive procedures with advice, and lower bounds on hardness amplification proofs". In: *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2018, pp. 956–966.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. "Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes". In: *Journal of the ACM* 56.4 (2009), Art. 20, 34.

[GV04] Dan Gutfreund and Emanuele Viola. "Fooling parity tests with parity gates". In: *Proc. 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2004, pp. 381–392.

[GW02] Oded Goldreich and Avi Wigderson. "Derandomization that is rarely wrong from short advice that is typically good". In: *Proc. 6th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2002, pp. 209–223.

[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 $\mathbf{NC}^1$". In: *Computational Complexity* 17.1 (2008), pp. 3–37.

[Hoz21] William M. Hoza. "Better pseudodistributions and derandomization for space-bounded computation". In: *Proc. 25th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2021, 28:1–28:23.

[HR03] Tzvika Hartman and Ran Raz. "On the distribution of the number of roots of polynomials and explicit weak designs". In: *Random Structures & Algorithms* 23.3 (2003), pp. 235–263.

[HV06] Alexander Healy and Emanuele Viola. "Constant-depth circuits for arithmetic in finite fields of characteristic two". In: *Proc. 23rd Symposium on Theoretical Aspects of Computer Science (STACS)*. 2006, pp. 672–683.

[IKO+08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Cryptography with constant computational overhead". In: *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*. 2008, pp. 433–442.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. "In search of an easy witness: exponential time vs. probabilistic polynomial time". In: *Journal of Computer and System Sciences* 65.4 (2002), pp. 672–694.

[IW01] Russel Impagliazzo and Avi Wigderson. "Randomness vs. time: Derandomization under a uniform assumption". In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 672–688.

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

[KI04] Valentine Kabanets and Russell Impagliazzo. "Derandomizing polynomial identity tests means proving circuit lower bounds". In: *Computational Complexity* 13.1-2 (2004), pp. 1–46.

[KM02] Adam 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.

[KMS12] Jeff Kinne, Dieter van Melkebeek, and Ronen Shaltiel. "Pseudorandom generators, typically-correct derandomization, and circuit lower bounds". In: *Computational Complexity* 21.1 (2012), pp. 3–61.

[Kou+16] Michal Koucký et al. "Catalytic computation". In: *Bulletin of the European Association for Theoretical Computer Science (EATCS)* 1.118 (2016).

[KT22] Itay Kalev and Amnon Ta-Shma. "Unbalanced expanders from multiplicity codes". In: *Proc. 26th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2022, 12:1–12:14.

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

[LP22b] Yanyi Liu and Rafael Pass. "Leakage-resilient hardness vs. randomness". In: *Electronic Colloquium on Computational Complexity: ECCC* 30 (2022), p. 113.

[MST06] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. "On $\varepsilon$-biased generators in $\mathbf{NC}^0$". In: *Random Structures & Algorithms* 29.1 (2006), pp. 56–81.

[MW18] Cody Murray and R. Ryan Williams. "Circuit lower bounds for nondeterministic quasi-polytime: An easy witness lemma for $\mathbf{NP}$ and $\mathbf{NQP}$". In: *Proc. 50th Annual ACM Symposium on Theory of Computing (STOC)*. 2018.

[Nis92] Noam Nisan. "Pseudorandom generators for space-bounded computation". In: *Combinatorica* 12.4 (1992), pp. 449–461.

[Nis94] Noam Nisan. "$\mathbf{RL} \subseteq \mathbf{SC}$". In: *Computational Complexity* 4.1 (1994), pp. 1–11.

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs. randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167.

[OST22] Igor C. Oliveira, Rahul Santhanam, and Roei Tell. "Expander-based cryptography meets natural proofs". In: *Computational Complexity* 31.1 (2022), pp. 1–60.

[OW14] Ryan O'Donnell and David Witmer. "Goldreich's PRG: evidence for near-optimal polynomial stretch". In: *Proc. 29th Annual IEEE Conference on Computational Complexity (CCC)*. 2014, pp. 1–12.

[RS21] Hanlin Ren and Rahul Santhanam. "Hardness of KT characterizes parallel cryptography". In: *Proc. 36th Annual IEEE Conference on Computational Complexity (CCC)*. 2021.

[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.

[Sav70] Walter J. Savitch. "Relationships between nondeterministic and deterministic tape complexities". In: *Journal of Computer and System Sciences* 4.2 (1970), pp. 177–192.

[Spi96] Daniel A. Spielman. "Linear-time encodable and decodable error-correcting codes". In: vol. 42. 6, part 1. Codes and complexity. 1996, pp. 1723–1731.

[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.

[SV08] Ronen Shaltiel and Emanuele Viola. "Hardness amplification proofs require majority". In: *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*. 2008, pp. 589–598.

[SZ99] Michael E. Saks and Shiyu Zhou. "$\mathbf{BP_HSPACE}(S) \subseteq \mathbf{DSPACE}(S^{2/3})$". In: *Journal of Computer and System Sciences* 58.2 (1999), pp. 376–403.

[Tel19] Roei Tell. "Proving that $\mathbf{prBPP} = \mathbf{P}$ is as hard as proving that "almost $\mathbf{NP}$" is not contained in $\mathbf{P/poly}$". In: *Information Processing Letters* 152 (2019), p. 105841.

[Tel20] Roei Tell. "On implications of better sub-exponential lower bounds for $\mathbf{AC}^0$". Manuscript. 2020. URL: https://sites.google.com/site/roeitell/Expositions.

[TV07] Luca Trevisan and Salil Vadhan. "Pseudorandomness and average-case complexity via uniform reductions". In: *Computational Complexity* 16.4 (2007), pp. 331–364.

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

[VDHS13] Joris Van Der Hoeven and Éric Schost. "Multi-point evaluation in higher dimensions". In: *Applicable Algebra in Engineering, Communication and Computing* 24.1 (2013), pp. 37–52.

[Vio03] Emanuele Viola. "Hardness vs. randomness within alternating time". In: *Proc. 18th Annual IEEE Conference on Computational Complexity (CCC)*. 2003, pp. 53–69.

[Wil11] R. Ryan Williams. "Non-uniform ACC circuit lower bounds". In: *Proc. 26th Annual IEEE Conference on Computational Complexity (CCC)*. 2011, pp. 115–125.

[Zuc97] David Zuckerman. "Randomness-optimal oblivious sampling". In: *Random Structures & Algorithms* 11.4 (1997), pp. 345–367.

# A The Impagliazzo–Wigderson Derandomized Direct Product

In this appendix we include a full proof of Theorem 4.5, for completeness. The proof is a fleshing out and an elaboration of the proof that already appears in the original work [IW97], while verifying that their construction has the properties that we need.

**Theorem A.1** (Theorem 4.5, restated)**.** *There exists a constant $c > 1$ such that for any two constants $\delta, \gamma > 0$, and every $\varepsilon \colon \mathbb{N} \to (0,1)$, the following holds. There exists a logspace-computable,*

$$\left(\varepsilon \to 1-\delta, Q = \left(\frac{\log N}{\varepsilon^2}\right)^c\right)$$

*locally approximately lost-decodable code*

$$\mathsf{IW} \colon \{0,1\}^N \to (\{0,1\}^k)^{\bar{N}} ,$$

*where $\bar{N} = (N/\varepsilon)^{c\cdot(1/\gamma+1/\delta^2)}$ and $k = (c/\delta^2)\cdot\log(1/\varepsilon)$, such that each local decoder $\mathsf{Dec}_i$ is a constant-depth oracle circuit of size $N^\gamma/\varepsilon^2$ that makes non-adaptive queries and has a top majority gate of fan-in $Q$.*

*Moreover, there exists a randomized oracle machine $R$ such that the following holds. On input $1^N$ and oracle access to $f \in \{0,1\}^N$, $R$ runs in space $\frac{c}{\gamma}\log N$, uses $\frac{c}{\varepsilon^2\gamma}\log N$ bits of randomness, makes at most $N^\gamma$ queries to $f$, and prints a constant-depth oracle circuit $C$ of size $N^\gamma/\varepsilon^2$ that makes non-adaptive queries and has a top majority gate of fan-in $Q$ such that the following holds. For any $\tilde{f} \in \Sigma^{\bar{N}}$ such that $\Pr_{z\in[\bar{N}]}[\tilde{f}_z = \mathsf{IW}(f)_z] \geq \varepsilon$, with probability at least $0.99$ it holds that $\Pr_{x\in[N]}\left[C^{\tilde{f}}(x) = f(x)\right] \geq 1-\delta$.*

**Proof.** We are given a string $f \in \{0,1\}^N$ and encode it to $\bar{f} \in \Sigma^{\bar{N}}$, where $\bar{N} = \mathrm{poly}(N)$ and $\Sigma = \{0,1\}^k$; the precise values of both $\bar{N}$ and $k$ will be specified below. We use the following two ingredients.

- The strong sampler $\mathsf{Samp}\colon \{0,1\}^{m_1} \times [k] \to \{0,1\}^n$ from Theorem 3.8, using parameters $n = \log N$ and $\eta = \delta/2$ and $\mu = \varepsilon/8$. With these parameters we have $k = O(\log(1/\varepsilon)/\delta^2)$ and $m_1 = n + O(k)$.
- The $(n,\rho)$-design $\mathsf{Des}\colon \{0,1\}^{m_2} \times [k] \to \{0,1\}^n$ from Theorem 3.11, set with $\alpha = \gamma/2$. By our choice of parameters, $\rho = 2^{(\gamma/2)n}$, $m_2 = O(n/\gamma)$ and $k$ can be as large as $2^{(\gamma/2c_k)n}$ for some universal constant $c_k$. Indeed, the latter fits the constraint regarding the sampler's degree, as $\delta$ and $\gamma$ are constants.

Let $\bar{n} = m_1 + m_2 = O(n/\gamma + \log(1/\varepsilon)/\delta^2) = O_{\delta,\gamma}(n + \log(1/\varepsilon))$ and $\bar{N} = 2^{\bar{n}}$. Given $\bar{z} = (z_1, z_2) \in \{0,1\}^{\bar{n}}$ and $i \in [k]$, we define

$$\mathsf{Loc}(\bar{z}, i) = \mathsf{Samp}(z_1, i) \oplus \mathsf{Des}(z_2, i) \in \{0,1\}^n \ .$$

Then, the code $\mathsf{IW}\colon \{0,1\}^N \to \left(\{0,1\}^k\right)^{\bar{N}}$ maps $f$ to $\bar{f} = \mathsf{IW}(f)$, where for every $\bar{z} \in [\bar{N}]$,

$$\bar{f}_{\bar{z}} = \left(f_{\mathsf{Loc}(\bar{z},1)}, \ldots, f_{\mathsf{Loc}(\bar{z},k)}\right) \ .$$

**Local list-decoding.** The proof follows the standard reconstruction approach: We show an oracle circuit that gets access to a "corrupted" codeword $\bar{C}$ (i.e., $\bar{C}$ computes a function that agrees with $\bar{f}$ on $\varepsilon$ fraction of the inputs) and computes $f$ correctly on $1-\delta$ of the inputs. The main step in the proof is showing the following lemma.

**Lemma A.2.** *There exists a randomized procedure $P$ that uses $\bar{n} - n + \log(k) + k + 1$ random coins, makes $(k-1) \cdot 2^{(\gamma/2)\cdot n}$ queries to $f$, and prints an oracle circuit $F_0\colon \{0,1\}^n \to \{0,1\}$ such that the following holds:*

1. *For any $\bar{C}\colon \{0,1\}^{\bar{n}} \to \Sigma$ computing $\bar{f}$ correctly on at least an $\varepsilon$ fraction of its inputs, for at least $1-\delta$ of the inputs $x \in \{0,1\}^n$ it holds that $\Pr[F_0^{\bar{C}}(x) = f_x] \geq 1/2 + \varepsilon/64$, where the probability is over the internal randomness of $P$.*
2. *The circuit $F_0$ is a constant-depth oracle circuit of size $O(k \cdot 2^{(\gamma/2)\cdot n})$ (over the De Morgan basis) making a single queries.*

Establishing the lemma will suffice to prove the theorem. To see this, let $\bar{P}$ be the procedure that runs $P$ for $Q = O(n/\varepsilon^2)$ times and prints a circuit $\mathsf{Dec}\colon \{0,1\}^n \to \{0,1\}$ that computes the majority vote among the circuits $F_0$ that were printed. Note that:

1. For every fixed $\bar{C}$ that agrees with $\bar{f}$ on at least $\varepsilon$ inputs, with high probability over the coins of $\bar{P}$ we have that $\mathsf{Dec}^{\bar{C}}$ succeeds in computing $f$ on $1-\delta$ of the inputs $x \in \{0,1\}^n$.

2. The circuit Dec has constant depth and size $O\left((n/\varepsilon^2)\cdot k\cdot 2^{(\gamma/2)\cdot n}\right) < 2^{\gamma\cdot n}/\varepsilon^2$, and it uses $Q$ non-adaptive oracle queries and a top majority gate of fan-in $Q$.

Denote by $\mathsf{Dec}_1,...,\mathsf{Dec}_L$ the circuits that are obtained by enumerating over the possible choices of $\bar{P}$. By the above, the circuits $\mathsf{Dec}_i$ have complexity as asserted in the theorem, and for every $\bar{C}$ that agrees with $\bar{f}$ on $\varepsilon$ of the inputs there is $i\in[L]$ such that $\mathsf{Dec}_i^{\bar{C}}$ computes $f$ correctly on $1-\delta$ of the inputs. The rest of the proof is therefore devoted to establishing Lemma A.2.

The randomized procedure $P$. We choose uniformly at random the following:

- A seed $z_1\in\{0,1\}^{m_1}$ for the sampler,
- An index $i\in[k]$, and,
- Values $\alpha\in\{0,1\}^{m_2-n}$ for the entries of $z_2\in\{0,1\}^{m_2}$ on the coordinates outside $S_i$ (i.e., outside the $i$-th set in the design that Des computes).

Recall that $F_0$ will get an input $x\in\{0,1\}^n$; given any such $x$, it is possible to combine it with $\alpha$ into $z_2\in\{0,1\}^{m_2}$ (i.e., $z_2{\restriction}_{S_i}=x$ and $z_2{\restriction}_{[m_2]\setminus S_i}=\alpha$). For each $j\in[k]$ different from $i$, note that when iterating over all inputs $x\in\{0,1\}^n$ to $F_0$ and combining $\alpha$ and $x$ into $z_2$, there are at most $2^{(\gamma/2)\cdot n}$ possible outputs for $\mathsf{Des}(z_2,j)$. Thus, when iterating over all possible $x$ and completing $x$ to a corresponding $\bar{z}=(z_1,z_2)$, there are at most $\rho=2^{(\gamma/2)\cdot n}$ possible values for $\mathsf{Loc}(\bar{z},j)$ given that $\alpha$, $i$, and $z_1$ are fixed.

The procedure $P$ queries $f$ at the $(k-1)\cdot 2^{(\gamma/2)\cdot n}$ locations $\mathsf{Loc}(\bar{z},j)$ above, and hard-wires into $F_0$ the choices of $(z_1,i,\alpha)$, the answers to the queries, the values

$$\mathsf{Samp}(z_1,1),\ldots,\mathsf{Samp}(z_1,k),$$

and the design underlying Des. Given $x\in\{0,1\}^n$, the circuit $F_0$:

1. Shifts $x$ to $x'=x\oplus\mathsf{Samp}(z_1,i)$, completes $x'$ (using $\alpha$) to $z_2\in\{0,1\}^{m_2}$, and queries $\bar{C}$ on input $\bar{z}=(z_1,z_2)$.

   (To parse the meaning of this step, note that $\mathsf{Loc}(\bar{z},i)=\mathsf{Samp}(z_1,i)\oplus x'=x$, so we hope to have $\bar{C}(\bar{z})_i=(\bar{f}_{\bar{z}})_i=f_x$.)

2. For each $j\in[k]$, $j\neq i$, let $c_j\in\{0,1\}$ equal zero iff $\bar{C}(\bar{z})_j=(\bar{f}_{\bar{z}})_j=f_{\mathsf{Loc}(\bar{z},j)}$.

3. For $\ell=\sum_{j\neq i}c_j$, output $\bar{C}(\bar{z})_i$ with probability $2^{-\ell}$ and a random bit otherwise.

Note that the description above is of a *probabilistic* circuit $F_0$, which uses random coins $r\in\{0,1\}^{k+1}$ in Step (3). However, we can choose these coins $r$ in advance and hard-wire them into $F_0$.

The complexity of $F_0$. The number of non-uniform advice bits for $F_0$ is $m_1+\log(k)+(m_2-n)+(k-1)\cdot 2^{(\gamma/2)\cdot n}+k\cdot n+k\cdot m_2=O(k\cdot 2^{(\gamma/2)\cdot n})$. Computationally, in Step (2) the circuit

$F_0$ needs to compute $\mathsf{Loc}(\bar{z}, j) = \mathsf{Samp}(z_1, j) \oplus \mathsf{Des}(z_2, j)$ for all $j \neq i$. This reduces to computing $\mathsf{Des}(z_2, j)$, which is trivial since we hard-wired the design.

To see how to compute Step (3), denote the sequence of random coins by $\bar{r} = r_1, ..., r_k$ and the sequence of bits computed in Step (2) by $\bar{c} = c_1, ..., c_k$. Then, for each $j \in [k] \setminus \{i\}$ let $d_j = 1 \iff (c_j = 0) \vee (c_j = 1 \wedge r_j = 1)$, and let $D = \wedge_{j \in [k] \setminus \{i\}} d_j$. Observe that over a random choice of $\bar{r}$, the probability that $D = 1$ is precisely $2^{-\sum_{j \neq i} c_j}$. Thus, Step (3) can be computed in constant depth over the De Morgan basis.

The complexity of $P$. We argue that $P$ can be implemented by a probabilistic algorithm using space $O(n/\gamma) = O(n)$. Towards doing so, we describe the execution of $P$ and the way it prints the circuit $F_0$ in more detail.

In the first step, $P$ randomly chooses $z_1, i, \alpha$ and stores them on its work tapes. This can be done using $O(n)$ space. Then, $P$ iterates over all choices for $j \in [k]$. For each $j$, it computes the set $S_i \cap S_j$ (using the fact that the mapping of $j \mapsto S_j$ can be done in space $O_\gamma(n)$; see Theorem 3.11) and stores it. Then, it iterates over all possible choices for $x^{(j)} \in \{0,1\}^{|S_i \cap S_j|}$, in lexicographical order. For every fixed $j$ and $x^{(j)}$, it computes the $n$-bit string $\ell$ that is obtained by placing $x^{(j)}$ in the locations $S_i \cap S_j$ and $\alpha|_{S_j \setminus S_i}$ in the locations $S_j \setminus S_i$, and XORing the result with $\mathsf{Samp}(z_1, j)$. Note that, by definition, $\ell = \mathsf{Loc}(\bar{z}, j)$, where $\bar{z} = (z_1, z_2)$ and $z_2$ depends on $x^{(j)}$ as in the first description of $P$ above.

For each choice of $(j, x^{(j)})$ and a corresponding $\mathsf{Loc}(\bar{z}, j)$, the algorithm $P$ queries $f$ at location $\mathsf{Loc}(\bar{z}, j)$, and prints to its output an additional gate in the description of the circuit: The index of this gate is $(j, x^{(j)}) \in [2^{\gamma \cdot n}]$, its type is the constant $f_{\mathsf{Loc}(\bar{z}, j)}$, and it has no gates feeding into it. Since there are less than $k \cdot 2^{\gamma \cdot n} < 2^{2n}$ choices for $j$ and $x^{(j)}$, this entire step can be done in space $O(n)$.

Next, the algorithm $P$ prints another set of $k \cdot m_2$ constant gates, which describe the sets $S_1, ..., S_k \subseteq [m_2]$ in the design (i.e., the description of each $S_i$ is the indicator vector of the $n$-bit set $S_i \subseteq [m_2]$). Then, $P$ prints yet another set of $k \cdot n$ constant gates, which describe the values of $\mathsf{Samp}(z_1, j)$ for the fixed $z_1$ chosen in advance and for all $j \in [k]$. This step can also be done in space $O(m_2) = O(n/\gamma)$, relying on Theorem 3.11 and Theorem 3.9.

Finally, the algorithm $P$ prints gates that implement the rest of the functionality of $F_0$. Specifically, it prints $n$ input gates, then prints gates that shift any input $x$ to $x' = x \oplus \mathsf{Samp}(z_1, i)$, and that complete $x'$ to $z_2$. Then, it prints gates that, in parallel, query the oracle (given to the circuit) in location $\bar{z}$, and for every $j \in [k]$, query the constant gates to obtain $f_{\mathsf{Loc}(\bar{z}, j)}$. (Note that the indices of these constant gates are computable by the circuit in constant depth, given that the values $\mathsf{Samp}(z_1, j)$ for all $j \in [k]$ and the sets $S_j, j \in [k]$ are also stored in constant gates in the circuit.) It then prints gates that for each $j \in [k]$ check whether $\bar{C}(\bar{z})_j = f_{\mathsf{Loc}}(\bar{z}, j)$, and gates that sum the results into an integer $\ell$. The last set of constant gates are $k + 1$ constants that $P$ chooses at random "on the fly", to serve as randomness $r$ for the circuit. The last set of gates in the circuit implement Step (3) in the functionality of $F_0$, and we already showed above (when describing the complexity of $F_0$) an explicit way to do so. Note that printing the entire functionality of the circuit never requires storing more than $O(n)$ bits on the work tapes of $P$, and thus overall the space complexity of $P$ is $O(n)$.

Correctness. Fix any $\bar{C}\colon \{0,1\}^{\bar{n}} \to \{0,1\}$ that agrees with $\bar{f}$ on at least $\varepsilon$ fraction of the inputs. We show that for any $H \subseteq \{0,1\}^n$ of density $|H| \geq \delta \cdot 2^n$ it holds that

$$\Pr_{i,z_1,\alpha,r,x\in H}\left[F_0(x) = f_x\right] \geq 1/2 + \varepsilon/64\ .$$

Establishing this claim sufices for the proof, since it implies that the set of "bad" inputs $\{x \in \{0,1\}^n : \mathbb{E}_{i,z_1,\alpha,r}\left[F_0(x) = f_x\right] < 1/2 + \varepsilon/64\}$ has density less than $\delta$.

We introduce additional notation. For any fixed $\bar{z} \in \{0,1\}^{\bar{n}}$, let

$$\overline{\mathsf{Loc}}(\bar{z}) = (\mathsf{Loc}(\bar{z},1),\ldots,\mathsf{Loc}(\bar{z},k)).$$

We denote by $\bar{\mathbf{z}}$ the uniform distribution over $\mathbf{u}_{\bar{n}}$. For any fixed $i \in [k]$, let $\bar{\mathbf{z}}^{|i\in H}$ denote the distribution $\bar{\mathbf{z}}$ conditioned on $\mathsf{Loc}(\bar{\mathbf{z}},i) \in H$. Finally, denote by $\bar{\mathbf{z}}^{|\mathbf{u}_{[k]}\in H}$ the distribution $\bar{\mathbf{z}}^{|i\in H}$ when $i \in [k]$ is chosen at random (i.e., first uniformly choose $i \in [k]$ and then draw a sample from $\mathbf{z}^{|i\in H}$).

We need the following claim that relies on the properties of the sampler:

**Claim A.3.** *The probability over $\bar{z} \sim \bar{\mathbf{z}}$ that $\Pr_{j\in[k]}\left[\mathsf{Loc}(\bar{z},j) \in H\right] < \delta/2$ is at most $\varepsilon/8$.*

*Proof.* For any fixed $z_2 \in \{0,1\}^{m_2}$, the distribution $\overline{\mathsf{Loc}}(\bar{\mathbf{z}})$ conditioned on the second part of $\bar{\mathbf{z}}$ being $z_2$ is

$$\overline{\mathsf{Loc}}(\mathbf{u}_{m_1}, z_2) = (\mathsf{Samp}(\mathbf{u}_{m_1})_1, ..., \mathsf{Samp}(\mathbf{u}_{m_1})_k) \oplus (\mathsf{Des}(z_2)_1, ..., \mathsf{Des}(z_2)_k)\ ,$$

where the $k$ mentions of $\mathbf{u}_{m_1}$ in the expression above all represent a single instance of the random variable (i.e., a single choice of $z_1 \sim \mathbf{u}_{m_1}$ that is reused $k$ times). For each fixed $z_2 \in \{0,1\}^{m_2}$ and $j \in [k]$, denote $H_j^{z_2} = \{h \oplus \mathsf{Des}(z_2)_j : h \in H\}$. By the properties of the sampler, for every $z_2$ we have

$$\Pr_{z_1\sim\mathbf{u}_{m_1}}\left[\left|\Pr_{j\in[k]}\left[\mathsf{Samp}(z_1,j) \in H_j^{z_2}\right] - \mathop{\mathbb{E}}_{j\in[k]}\left[\rho(H_j^{z_2})\right]\right| \leq \frac{\delta}{2}\right] \geq \frac{7\varepsilon}{8},$$

so with probability at least $7\varepsilon/8$ over $z_1 \sim \mathbf{u}_{m_1}$ we have $\Pr_{j\in[k]}\left[\mathsf{Samp}(z_1,j) \in H_j^{z_2}\right] \geq \delta/2$, which readily implies that $\Pr_{i\in[k]}[\mathsf{Loc}((z_1,z_2),j) \in H] \geq \delta/2$. As this is true for any $z_2$, the claim follows. □

Relying on Claim A.3, we now argue the following.

**Lemma A.4.** *There exists $\bar{X} \subseteq (\{0,1\}^n)^k$ such that $\Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|\mathbf{u}_{[k]}\in H}) \in \bar{X}\right] = \varepsilon/4$, and for any $\bar{z}$ such that $\overline{\mathsf{Loc}}(\bar{z}) \in \bar{X}$ it holds that $\bar{C}(\bar{z}) = \bar{f}_{\bar{z}}$.*

*Proof.* For any fixed $\bar{x} = (x_1, ..., x_k) \in (\{0,1\}^n)^k$, let $h(\bar{x}) = |\{i \in [k] : x_i \in H\}|$. Then, we have that

$$\begin{aligned}
\Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|\mathbf{u}_{[k]} \in H}) = \bar{x}\right] &= \mathop{\mathbb{E}}_{i \in [k]}\left[\Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|i \in H}) = \bar{x}\right]\right] \\
&= \mathop{\mathbb{E}}_{i \in [k]}\left[\frac{\Pr[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}) = \bar{x} \wedge x_i \in H]}{\Pr[\mathsf{Loc}(\bar{\mathbf{z}}, i) \in H]}\right] \\
&\geq (1/\delta) \cdot \mathop{\mathbb{E}}_{i \in [k]}\left[\Pr[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}) = \bar{x} \wedge x_i \in H]\right] \\
&= (h(\bar{x})/\delta k) \cdot \Pr[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}) = \bar{x}] \,. \qquad \text{(A.1)}
\end{aligned}$$

By Claim A.3 we have that $\Pr[h(\overline{\mathsf{Loc}}(\bar{\mathbf{z}})) < (\delta/2) \cdot k] < \varepsilon/8$, and by a union-bound,

$$\Pr_{\bar{z} \sim \bar{\mathbf{z}}}\left[\bar{C}(\bar{z}) \neq \bar{f}_{\bar{z}} \vee h(\bar{x}) < \frac{\delta}{2} \cdot k\right] \leq (1 - \varepsilon) + \frac{\varepsilon}{8} = 1 - \frac{7\varepsilon}{8} \,.$$

Denote by $\bar{Z} \subseteq \{0,1\}^{\bar{n}}$ the set of $\bar{z}$-s such that $\bar{C}(\bar{z}) = \bar{f}_{\bar{z}}$ and $h(\bar{x}) \geq (\delta/2) \cdot k$, and denote $\bar{X} = \left\{\overline{\mathsf{Loc}}(\bar{z}) : \bar{z} \in \bar{Z}\right\}$. By Equation (A.1),

$$\Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|\mathbf{u}_{[k]} \in H}) \in \bar{X}\right] \geq \frac{1}{2} \cdot \Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}) \in \bar{X}\right] > \frac{\varepsilon}{4} \,.$$

Finally, we take $\bar{X}$ to be a subset of the above set that has weight exactly $\varepsilon/4$ (rather than at least $\varepsilon/4$) under $\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|\mathbf{u}_{[k]} \in H})$. □

We now consider a choice of $(i, z_1, \alpha, r)$ as before for constructing $F_0$, and a random choice of input $x \in H$ for $F_0$. For a given choice of $(i, z_1, \alpha, x)$ (that will typically be clear from context), we denote by $\bar{z} = \bar{z}(i, z_1, \alpha, x)$ the string that is obtained by combining $(i, z_1, \alpha, x)$ into $\bar{z}$ in the same way $F_0$ combines them (i.e., the string obtained by shifting $x$ to $x' = x \oplus \mathsf{Samp}(z_1, i)$, completing $x'$ to $z_2$ using $\alpha$, and concatenating $\bar{z} = (z_1, z_2)$). Then, we have that

$$\begin{aligned}
&\Pr_{i,z_1,\alpha,r,x \in H}\left[F_0^{\bar{C}}(x) = f_x\right] \\
&= \Pr_{i,z_1,\alpha,x \in H}\left[\overline{\mathsf{Loc}}(\bar{z}) \in \bar{X}\right] \cdot \mathop{\mathbb{E}}_{i,z_1,\alpha,x \in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \in \bar{X}\right] \\
&\quad + \Pr_{i,z_1,\alpha,x \in H}\left[\overline{\mathsf{Loc}}(\bar{z}) \notin \bar{X}\right] \cdot \mathop{\mathbb{E}}_{i,z_1,\alpha,x \in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \notin \bar{X}\right] \\
&= \Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|\mathbf{u}_{[k]} \in H}) \in \bar{X}\right] \cdot \mathop{\mathbb{E}}_{i,z_1,\alpha,x \in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \in \bar{X}\right] \\
&\quad + \Pr\left[\overline{\mathsf{Loc}}(\bar{\mathbf{z}}^{|\mathbf{u}_{[k]} \in H}) \notin \bar{X}\right] \cdot \mathop{\mathbb{E}}_{i,z_1,\alpha,x \in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \notin \bar{X}\right] \,, \qquad \text{(A.2)}
\end{aligned}$$

where the last equality is a bit subtle. To see that it holds, first note that a choice of $\bar{z} = \bar{z}(i, z_1, \alpha, x)$ according to a uniform choice of $(i, z_1, \alpha, x \in H)$ can be thought of as uniformly choosing $z_1 \in \{0,1\}^{m_1}$, $i \in [k]$, and $x \in H$, then putting $x' = x \oplus \mathsf{Samp}(z_1)_i$

in the positions of the $i$-th subset in $z_2$, and finally completing the rest of the positions in $z_2$ according to $\alpha$ and concatenating $\bar{z} = (z_1, z_2)$. This process, in turn, is equivalent to a uniform choice of $\bar{z} \sim \bar{\mathbf{z}}$ conditioned on $\mathsf{Loc}(\bar{z})_i \in H$ for a random $i \in [k]$.

Now, plugging Lemma A.4 into Equation (A.2), we have that

$$\Pr_{i,z_1,\alpha,r,x\in H}\left[F_0^{\bar{C}}(x) = f_x\right] \geq \frac{\varepsilon}{4} + \left(1 - \frac{\varepsilon}{4}\right) \cdot \mathop{\mathbb{E}}_{i,z_1,\alpha,x\in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \notin \bar{X}\right]. \quad \text{(A.3)}$$

To further lower-bound the RHS in Equation (A.3), denote by $\mathcal{H}$ the event

$$\Pr_{j\in[k]}\left[\mathsf{Loc}(\bar{z}, j) \in H\right] \geq \delta/2,$$

and note that

$$\begin{aligned}
&\mathop{\mathbb{E}}_{i,z_1,\alpha,x\in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \notin \bar{X}\right] \\
&\geq \left(1 - (1+2\varepsilon)\frac{\varepsilon}{8}\right) \cdot \mathop{\mathbb{E}}_{i,z_1,\alpha,x\in H}\left[\Pr_r\left[F_0^{\bar{C}}(x) = f_x\right] | \overline{\mathsf{Loc}}(\bar{z}) \notin \bar{X}, \mathcal{H}\right] \qquad \text{(see below)} \\
&= \left(1 - (1+2\varepsilon)\frac{\varepsilon}{8}\right) \cdot \mathop{\mathbb{E}}_{\hat{z}\sim\widehat{\mathbf{z}}}\left[\Pr_{i\in\{j\in[k]:\bar{x}_j\in H\},z_1,\alpha,r}\left[F_0^{\bar{C}}(\bar{x}_i) = f_{\bar{x}_i} | \bar{z}(i, z_1, \alpha, \overline{\mathsf{Loc}}(\hat{z}, i)) = \hat{z}\right]\right], \quad \text{(A.4)}
\end{aligned}$$

where $\widehat{\mathbf{z}}$ is the distribution obtained by choosing $(i, z_1, \alpha, x \in H)$ conditioned on the event $\bar{z}(i, z_1, \alpha, x) \notin \bar{X}$ and on $\mathcal{H}$.[29] Now, to see that the inequality above is true, recall that by Claim A.3 we have that $\Pr_{\bar{z}\sim\bar{\mathbf{z}}}[\mathcal{H}] \geq 1 - \varepsilon/8$. The choice of $\bar{z}$ according to $(i, z_1, \alpha, x \in H)$ is not uniform, but it is equivalent to a uniform choice conditioned on some location of $\bar{x}$ being in $H$, and on $\bar{x} \notin \bar{X}$; these two events happen with probability at least $1 - \varepsilon/4 - \varepsilon/8 \geq 1 - \varepsilon$, conditioning on them can increase the probability of the bad event $\neg\mathcal{H}$ by a multiplicative factor of at most $1 + 2\varepsilon$ for $\varepsilon \leq 1/2$, which we can assume without loss of generality.

The final claim in the proof will allow us to lower bound $\Pr[F_0^{\bar{C}}(\bar{x}_i) = f_{\bar{x}_i} | \overline{\mathsf{Loc}}(\bar{z}) = \bar{x}]$ in Equation (A.4) for any fixed $\bar{x}$ for which $\mathcal{H}$ holds. Specifically:

**Lemma A.5.** *For every fixed $\hat{z} \in \{0,1\}^{\bar{n}}$ and $\bar{x} = \overline{\mathsf{Loc}}(\hat{z})$ it holds that*

$$\Pr_{i\in\{j\in[k]:\bar{x}_j\in H\},z_1,\alpha,r}\left[F_0^{\bar{C}}(\bar{x}_i) = f_{\bar{x}_i} | \bar{z}(i, z_1, \alpha, \overline{\mathsf{Loc}}(\hat{z}, i)) = \hat{z}\right] \geq \frac{1}{2} - 2^{-|I|/3},$$

*where $I = \{j \in [k] : \bar{x}_j \in H\}$.*

**Proof.** By the definition of $F_0$, when we condition on $\bar{z}(i, z_1, \alpha, \overline{\mathsf{Loc}}(\hat{z}, i)) = \hat{z}$, the output of $F_0^{\bar{C}}$ depends only on the choices of $i$ and of $r$. For every $j \in I$, let $c_j \in \{0,1\}$ equal zero

[29] Indeed, the equality relies on the fact that the choice of $(i, z_1, \alpha, x \in H)$ conditioned on $\bar{x} \notin \bar{X}$ and on $\mathcal{H}$ induces a probability distribution over $(\bar{x}, i)$, and that conditioning on a fixed $\bar{x}$ in this distribution yields a uniform choice of $i \in \{j \in [k] : \bar{x}_j \in H\}$.

iff $\bar{C}(\hat{z})_j = f_{\mathsf{Loc}(\hat{z},j)}$. Finally, let $\bar{\ell}_I = \sum_{j\in I} c_j$ and $\ell = \sum_{j\in[k]\setminus\{i\}} c_i$ (note that in $\bar{\ell}_I$ we are also counting $c_i$ at the "chosen" index $i$). Then, we have that

$$\begin{aligned}
\Pr_{i\in I,r}\left[F_0^{\bar{C}}(x) = f_x\right] &= \Pr_i\left[c_i = 1\right]\cdot\Pr_{r,i}\left[F_0^{\bar{C}}(x) = f_x | c_i = 1\right] + \Pr_i\left[c_i = 0\right]\cdot\Pr_{r,i}\left[F_0^{\bar{C}}(x) = f_x | c_i = 0\right] \\
&= \Pr_i\left[c_i = 1\right]\cdot\Pr_{r,i}\left[F_0^{\bar{C}}(x) \neq \bar{C}(\hat{z})_i\right] + \Pr_i\left[c_i = 0\right]\cdot\Pr_{r,i}\left[F_0^{\bar{C}}(x) = \bar{C}(\hat{z})_i | c_i = 0\right] \\
&= \frac{\bar{\ell}_I}{|I|}\cdot\frac{1-2^{-\ell+1}}{2} + \left(1-\frac{\bar{\ell}_I}{|I|}\right)\cdot\left(2^{-\ell}+\frac{1-2^{-\ell}}{2}\right) \\
&= \frac{1}{2}\cdot\left(1+2^{-\ell}\cdot\left(1-3\frac{\bar{\ell}_I}{|I|}\right)\right) \\
&\geq \frac{1}{2} - 2^{-|I|/3}\,,
\end{aligned}$$

where the last inequality follows by a case-analysis of the two cases $\bar{\ell}_I \leq |I|/3$ and $\bar{\ell}_I > |I|/3$ (in the latter case we have $\ell \geq \bar{\ell}_I > |I|/3$). ■

Plugging the above claim into Equation (A.4) and plugging the latter into Equation (A.3), and recalling the definition of $\mathcal{H}$, we have that

$$\begin{aligned}
\Pr_{i,z_1,\alpha,r,x\in H}\left[F_0^{\bar{C}}(x) = f_x\right] &\geq \frac{\varepsilon}{4} + \left(1-\frac{\varepsilon}{4}\right)\cdot\left(1-(1+2\varepsilon)\frac{\varepsilon}{8}\right)\cdot(1/2 - 2^{-\delta k/6}) \\
&> \frac{\varepsilon}{4} + \left(1-\frac{7\varepsilon}{16}\right)\cdot\left(\frac{1}{2} - 2^{-\delta k/6}\right) \\
&> \frac{1}{2} + \frac{\varepsilon}{64}\,,
\end{aligned}$$

where again, we assumed $\varepsilon$ is bounded from above by a small enough constant, and the last inequality relies on the fact that $k = \mathrm{poly}(\log(1/\varepsilon)/\delta)$.

Finally, we argue that $\mathsf{IW}$ is computable in logspace. Indeed, given $f \in \{0,1\}^N$, recall that each symbol $\bar{z} \in [\bar{N}]$ in $\bar{f} = \mathsf{IW}(f)$ is given by

$$\bar{f}_{\bar{z}} = \left(f_{\mathsf{Loc}(\bar{z},1)}, \ldots, f_{\mathsf{Loc}(\bar{z},k)}\right)$$

where $\mathsf{Loc}(\bar{z} = (z_1, z_2), i) = \mathsf{Samp}(z_1, i) \oplus \mathsf{Des}(z_2, i)$. We know that $\mathsf{Samp}(z_1, i)$ is computable in space $O(m_1)$ and $\mathsf{Des}(z_2, i)$ is computable in space $O(m_2)$. Altogether, $\bar{f}$ is computable in space

$$O(\log k + \log \bar{N} + m_1 + m_2) = O(n + \log(1/\varepsilon)) = O(\log \bar{N}),$$

as desired. ■

## A.1 A direct product theorem for smaller values of $\delta$

As long as we want to maintain a polynomially small rate, the above code of Theorem 4.5 can support a constant, or slightly sub-constant $\delta$. To see why this is the case, note that

while the random walks sampler of Theorem 3.8 has an optimal number of samples (i.e., $k = O(\log(1/\varepsilon)/\delta^2)$), its randomness complexity, namely $m_1 = n + O(k)$, is relatively high. Working out the parameters for a non-constant $\delta$, we get $\bar{N} = (1/\varepsilon)^{O(1/\delta^2)} \cdot \mathrm{poly}(N)$.

Using the samplers of Theorem 3.9 instead of those coming from random walks, we get better randomness complexity, at the expense of slightly more samples. In terms of parameters, $k$ becomes $\mathrm{poly}(\log(1/\varepsilon), 1/\delta)$, but now $m_1 = n + O(\log(1/\varepsilon\delta))$. Thus, we can get approximate local list decodability, where the "approximate" parameter $\delta$ is small.

Although we will not use this fact, it may be of independent interest and we state it here; the proof is nearly identical to the one above, where the only difference being using a different sampler and adjusting the number of sets in the design.

**Theorem A.6** (the derandomized direct product code of [IW97], instantiated with different samplers)**.** *There exists a constant $c > 1$ such that for any constant $\gamma > 0$, and every $\varepsilon, \delta \colon \mathbb{N} \to (0,1)$ such that $\delta(N) \geq \log(1/\varepsilon(N))N^{-c\gamma}$, the following holds. There exists a logspace-computable,*

$$\left(\varepsilon \to 1 - \delta, Q = \mathrm{poly}\left(\frac{\log N}{\varepsilon}\right)\right)$$

*locally approximately list-decodable code*

$$\mathsf{IW} \colon \{0,1\}^N \to (\{0,1\}^k)^{\bar{N}}$$

*where $\bar{N} = \mathrm{poly}(N/\varepsilon)$ and $k = \mathrm{poly}(\log(1/\varepsilon), 1/\delta)$, such that each local decoder $\mathsf{Dec}_i$ is a constant-depth oracle circuit of size $N^\gamma/\varepsilon^2$ that makes non-adaptive queries and has a top majority gate of fan-in $Q$.*

Note that compared to Theorem 4.5, the "symbol length" $k$ has worse dependence on $\varepsilon$, namely $\mathrm{poly}(\log(1/\varepsilon))$ instead of $O(\log(1/\varepsilon))$.[30]

# B Deferred Proofs of Technical Statements

## B.1 The uniform complexity of GGHKR′

We prove that GGHKR′ is logspace computable, following the notation of Theorem 4.4. We note that in [GGH+07], the encoding comprises two parts: The first part is an encoding of an error-correcting code, whereas the second part is a truth table of an $\mathbf{NC}^1$-complete language with nice properties (or more accurately, a "randomized image" of one), following [Bar89]. The second part is used, in [GGH+07], to reduce $\mathbf{NC}^1$ local decoding to an $\mathbf{AC}^0$

[30] We remark that going forward, using Theorem A.6 instead of Theorem 4.5 would have created an obstacle for us since we will eventually use a Hadamard encoding with blocklength $2^k$. However, a different, more efficient, inner code would improve the parameters, possibly at the expense of less efficient decoding. We do not delve into these trade-offs here.

local decoding. In Appendix B.2 we will establish $\mathbf{TC}^0$ local decoding (albeit by larger circuits) without resorting to [Bar89], so we can dispense with the second part.[31]

Given $x \in \{0,1\}^k$, our encoding returns $\mathsf{GGHKR}'(x) \in \{0,1\}^{k'}$ as follows.

- For an appropriate choice of parameters, we encode $x \in \{0,1\}^k$ into $x^{(1)} \in \mathbb{F}^{|\mathbb{F}|^m}$ via the low-degree extension view of the Reed–Muller code. Specifically, for $|\mathbb{F}| = \log^2 k$, $H = \log k$, and $m = \frac{\log k}{\log H}$, we interpret $x$ as an $m$-variate polynomial $\mathbb{F}^m \to \mathbb{F}$ of individual degree at most $H$ over a field of size $|\mathbb{F}|$, and output its evaluations over $\mathbb{F}^m$.

  Note that $x^{(1)} \in \mathbb{F}^{k_1}$ where $k_1 = k^2$, and that each symbol in $x^{(1)}$ can be computed from $x$ in space $O(\log|\mathbb{F}| + \log(H^m)) = O(\log k)$ via multivariate polynomial interpolation (see, e.g., [VDHS13]). We denote by $\mathcal{C}_1 \colon \{0,1\}^k \to \mathbb{F}^{k_1}$ the corresponding mapping.

- We employ an ABNNR-like distance amplification step [ABN+92] to map $x^{(1)} \in \mathbb{F}^{k_1}$ into $x^{(2)} \in (\mathbb{F}^d)^{k_1}$ by aggregating symbols according to a bipartite expander of degree $d = \mathrm{poly}(|\mathbb{F}|)$. Appropriate expanders exist with strongly explicit neighbors function (we will use ones with $\mathbf{AC}^0$ neighbor functions, see Appendix B.2), and in particular each symbol of $x^{(2)}$ can be computed from $x^{(1)}$ in space $O(\log(k_1 \log|\mathbb{F}|) + \log d + \log|\mathbb{F}|) = O(\log k)$.

  We let $\mathcal{C}_2 \colon \mathbb{F}^{k_1} \to (\mathbb{F}^d)^{k_1}$ denote the distance-amplification mapping.

We denote by $\mathcal{C}' \colon \{0,1\}^k \to \Sigma^{k_1}$ the composition $\mathcal{C}_2 \circ \mathcal{C}_1$, where $\Sigma = \mathbb{F}^d$ and $|\Sigma| = |\mathbb{F}|^d = 2^{\mathrm{polylog}(k)}$.

- Next, we concatenate $\mathcal{C}'$ with itself (with the appropriate parameters). Namely, let

$$\mathcal{C}'_{(1)} \colon \{0,1\}^k \to \Sigma^{k^2}$$

  be the code $\mathcal{C}'$ set with message length $k$ as above, and let

$$\mathcal{C}'_{(2)} \colon \{0,1\}^{\log|\Sigma|} \to \overline{\Sigma}^{\log^2|\Sigma|}$$

  be the code $\mathcal{C}'$ set with message length $\log|\Sigma|$, where $|\overline{\Sigma}| = 2^{\mathrm{polylog}(\log|\Sigma|)} = 2^{\mathrm{polyloglog}(k)}$. Then, we let $\mathcal{C}''$ be the code concatenation of $\mathcal{C}'_{(1)}$ and $\mathcal{C}'_{(2)}$. That is, $\mathcal{C}''$ gets $x \in \{0,1\}^k$, computes $x^{(2)} \in \Sigma^{k^2}$, and outputs $x'' \in (\bar{\Sigma}^{\log^2 \Sigma})^{k^2}$, wherein for all $(i,j) \in [k^2] \times [\log^2|\Sigma|]$, $x''_{(i,j)} = \mathcal{C}'_{(2)}(x_i^{(2)})_j$. Clearly,

$$\mathcal{C}'' \colon \{0,1\}^k \to \overline{\Sigma}^{k^2 \cdot \log^2|\Sigma|} = \overline{\Sigma}^{\widetilde{O}(k^2)}.$$

  By composition of space-bounded functions, both $\mathcal{C}'$ and $\mathcal{C}''$ are computable in space $O(\log k)$.

[31] We could follow [GGH+07] and get an $\mathbf{AC}^0$ circuit, however their technique seems to hinder the non-adaptivity of the local decoding.

Thus, $x'' = \mathcal{C}'_{(1)}(x) \in (\mathbb{F}^d)^{k_1}$ above is mapped to $x''' \in \overline{\Sigma}^{\widetilde{O}(k^2)}$ by the concatenation with $\mathcal{C}'_{(2)}$.

- Finally, we map $x''$ to the binary $x''' \in \{0,1\}^{k'}$ by another code concatenation, specifically encoding each symbol in $\overline{\Sigma} \equiv \{0,1\}^{\mathrm{polyloglog}(k)}$ by the Reed–Muller code (in its low-degree extension variant) concatenated with Hadamard [STV01], denoted by $\mathcal{C}_3 \colon \overline{\Sigma} \to \{0,1\}^{\mathrm{polyloglog}(k)}$. As the block length of $\mathcal{C}_3$ is very small, a naive implementation of each invocation of $\mathcal{C}_3$ takes $\mathrm{polyloglog}(k)$ space.

Overall, using space-efficient composition (see Proposition 3.2), we get that $\mathsf{GGHKR}'(x)$ is computable in space $O(\log k)$.

We note that [GGH+07] *does not* use $\mathcal{C}''$ before transforming to binary, but rather concatenates $\mathcal{C}'$ with $\mathcal{C}_3$. We choose to make an additional concatenation step in order to further reduce the message length of $\mathcal{C}_3$ (for a reason that will become apparent soon in Appendix B.2). In terms of parameters, due to the additional concatenation, one needs to set $\mathcal{C}'$ to decode from a larger (yet still constant) distance, compared to the parameters used in [GGH+07]. Inspecting their proof, this can easily be achieved by taking the expander's degree $d$ to be larger by a constant factor.[32]

## B.2 The uniform decoding of $\mathsf{GGHKR}'$

We now show that our variant of $\mathsf{GGHKR}'$ is locally decodable by a uniform local decoder, in the sense that there exists a small-space oracle machine that outputs the description of the (randomized) decoding circuit. (The correctness of the local decoding itself will mostly follow [GGH+07], up to the additional concatenation step which we will specifically address.)

**Claim B.1** (uniform decoding of $\mathsf{GGHKR}'$)**.** *The code* $\mathsf{GGHKR}'$ *of Theorem 4.4 is locally decodable by logspace uniform* $\mathbf{TC}^0$ *circuits. Namely, there is a deterministic Turing machine that runs in space* $\mathrm{polyloglog}(k)$ *and outputs a randomized* $\mathbf{TC}^0$ *oracle circuit* $D'$ *of size* $2^{\mathrm{polyloglog}(k)}$ *that locally uniquely decodes* $\mathsf{GGHKR}' \colon \{0,1\}^k \to \{0,1\}^{k'}$ *with non-adaptive queries, from* $\rho = \frac{24}{25}$ *fraction of agreement and with success probability at least* $1 - \frac{1}{k}$ *(over the circuit's internal randomness).*

**Proof.** We start by setting some notations, and follow Appendix B.1. For encoding

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

let $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}'$, $\mathcal{C}''$, and $\mathcal{C}_3$ as above. We describe the generation of a $\mathbf{TC}^0$ decoding circuit for $\mathsf{GGHKR}'$.

[32] Their code description and analysis can be found in the extended version of [GGH+07] titled "List Decoding in Constant Depth".

**The code $\mathcal{C}_1$.** For $\mathcal{C}_1$, following [GGH+07], we need to locally (uniquely) decode from very small distance, say $\delta_1 = \frac{1}{100|\mathbb{F}|}$. Recall that in local decoding of such RM codes, we pass a random line through the desired point and query the coordinates (or some of them) along the function's restriction to that line. In the small distance regime, with high probability (over the choice of the line), we only query points that agree with the low-degree polynomial. Thus, to locally decode, [GGH+07] apply standard interpolation. Outputting an oracle circuit that performs Lagrange interpolation can be done in logarithmic space, noting that constant-depth threshold circuits for field arithmetics are logspace-uniform (see, e.g., [HV06]). Thus, the local decoder circuit for $\mathcal{C}_1$ is a $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(|\mathbb{F}|) = \mathrm{polylog}(k)$ and can be generated in space $O(\log |\mathbb{F}|) = O(\log \log k)$. (The non-adaptivity readily follows from the non-adaptivity of the interpolation.)

**The code $\mathcal{C}_2$.** The decoding algorithm for $\mathcal{C}_2$ goes as follows. Let $\Gamma\colon [k_1]\times[d] \to [k_1]$ be the expander used for the symbol aggregation. As observed in [GGH+07] (following [GV04]), appropriate expanders exist with strongly explicit neighbours function, and in particular we can generate in space $O(\log \log k)$ an $\mathbf{AC}^0$ circuit of size $\mathrm{polylog}(k)$ that computes

$$\Gamma(v,1),\ldots,\Gamma(v,d)$$

given any $v \in [k_1]$. On oracle access to a word $y \in (\mathbb{F}^d)^{k_1}$ for which there exists $x \in \mathbb{F}^{k_1}$ that satisfies $\Delta(y, \mathcal{C}_2(x)) \leq \beta_1 k_1$ for some predetermined $\beta_1$, and an index $i \in [k_1]$, the decoder queries $y$ at locations $\Gamma(i,1),\ldots,\Gamma(i,d)$. Then, its guess for $x_i$ is the prediction that appears most often in the $d$ queries.[33] The latter can be done by a non-adaptive $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(d) = \mathrm{polylog}(k)$.[34] Overall, the decoding circuit is a non-adaptive $\mathbf{TC}^0$ circuit of size $\mathrm{polylog}(k)$. To generate the circuit, once we have the circuit for $\Gamma$, we can output the decoding circuit in additional $O(\log |\mathbb{F}|) = O(\log \log k)$ space.

**The code $\mathcal{C}'$.** 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. All queries are done in parallel, so we get a $\mathbf{TC}^0$ oracle circuit of size $\mathrm{polylog}(k)$. By simple manipulations and composition of space-bounded functions, the local decoder for $\mathcal{C}'$ can be printed in $O(\log \log k)$ space.

**The code $\mathcal{C}''$.** We need $\mathcal{C}''$ to be locally decodable from constant relative distance, say $\beta = 1/5$. Towards this end, we set the decoding distance of $\mathcal{C}'_{(1)}$ and $\mathcal{C}'_{(2)}$ accordingly, say $\sqrt{\beta}$ for

[33] In more detail, note that for each $j \in [d]$, every $z(j) = y_{\Gamma(i,j)}$ is a $d$-tuple. Thus, if $i$ is the $j'$-th left-neighbor of $\Gamma(i,j)$ then we take $z(j)_{j'}$ to be the $j$-th candidate for $x_i$. The prediction for $x_i$ is then the candidate that appears most often out of the $d$ candidates.

[34] To see this, one can use iterated integer addition (which is in logspace-uniform $\mathbf{TC}^0$) to compute the number of occurrences of each element, followed by standard comparisons which can be done in $\mathbf{AC}^0$.

both. Then, to get a local decoder for $\mathcal{C}''$, we employ standard decoding of concatenated codes. Namely, we first run the decoder for $\mathcal{C}'_{(1)}$ to compute the query locations. For every such query, we use the decoder for $\mathcal{C}'_2$ to decode the relevant $\Sigma$-symbol by going over all $\log|\Sigma|$ locations. All queries are done in parallel, so the resulting decoder is a non-adaptive $\mathbf{TC}^0$ circuit of size

$$\mathrm{polylog}(k) + \mathrm{polylog}(k) \cdot \log|\Sigma| \cdot \mathrm{polylog}(\log|\Sigma|) = \mathrm{polylog}(k),$$

where we used $\mathrm{polylog}(k)$ as an upper bound on the number of queries made by the decoder for $\mathcal{C}'_{(1)}$, and $\mathrm{polylog}(\log|\Sigma|)$ as an upper bound on the size of the decoder for $\mathcal{C}'_{(2)}$. The correctness follows from a standard averaging argument, and is given, say, in [Vad12, Problem 5.2].[35] In terms of uniformity, again we can generate the local decoder for $\mathcal{C}''$ in $O(\log\log k)$ space.

**The code $\mathcal{C}_3$.** Recall that $\mathcal{C}_3$ is the STV code [STV01] that maps $\overline{\Sigma} \equiv \{0,1\}^{\mathrm{polyloglog}(k)}$ into a binary string of length $\mathrm{polyloglog}(k)$. We don't need local decoding here, since the block-length is very small and the most naive decoding will suffice: To uniquely decode (from constant distance), we just go over all messages and check them one by one. Specifically, for all $z \in \overline{\Sigma}$, let $D_z$ be the circuit that has $\mathcal{C}(z)$ hard-coded, gets oracle access to some $y$ (of length $\mathrm{polyloglog}(k)$) and returns the Hamming distance between $y$ and $\mathcal{C}(z)$. Each $D_z$ is a (multi output-bit) $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(\log\log k)$ that can be generated in $\mathrm{polylog}(\log\log k) = O(\log\log k)$ space (which is the space required to compute the encoding $\mathcal{C}_3$, see [CT21a]). Now, the decoder for $\mathcal{C}_3$, given access to $y$, simply needs to choose the $z$ for which $D_z$ gives the maximal value. This can be implemented by a non-adaptive $\mathbf{TC}^0$ circuit of size $\mathrm{poly}(|\overline{\Sigma}|) = 2^{\mathrm{polyloglog}(k)}$ that can be generated in space $\mathrm{polyloglog}(k)$.

**Concatenating $\mathcal{C}''$ and $\mathcal{C}_3$.** Finally, recall that we apply code concatenation to get our final code $\mathsf{GGHKR}'$. The analysis here goes along the line of the analysis of $\mathcal{C}''$ above (and in fact is almost the same as in [GGH+07]) so we omit it. Overall, the code $\mathsf{GGHKR}'$ can be locally decodable by non-adaptive $\mathbf{TC}^0$ circuits of size $2^{\mathrm{polyloglog}(k)}$ that can be generated in space $\mathrm{polyloglog}(k)$.

The correctness of the decoding itself is established in [GGH+07] (up to the slight change of parameters needed in order to support the additional concatenation step). We also omitted the success probability analysis, which is standard (and closely follows [GGH+07]). ■

We conclude with two alternative approaches to modifying $\mathsf{GGHKR}'$ and its decoding. Those would not change our main result, and only affect Proposition 6.2.

[35] The only non-standard part is that we need to reduce the error of the decoder for $\mathcal{C}'_2$ so that whenever a block of length $\log(\Sigma)$ is of distance at most $\sqrt{\beta}$ from the correct symbol, the decoder succeeds in decoding each symbol of the block with probability at least (say) $\frac{1}{10Q\cdot\log(|\Sigma|)}$, where $Q \leq \mathrm{polylog}(k)$ is the query complexity of $\mathcal{C}'_1$. This increases the size complexity of the decoder for $\mathcal{C}'_2$ by $\mathrm{polyloglog}(k)$, which is insignificant in our setting.

- By concatenating with $\mathcal{C}'$ three times instead of twice, we can get the overall decoder size to $\mathrm{polylog}(k)$, generated in space $O(\mathrm{loglog} k)$.
- Similarly to [GGH+07], we can simulate the $\mathbf{TC}^0$ decoder by an $\mathbf{AC}^0$ decoder using an $\mathbf{NC}^1$-complete problem, in the spirit of Barrington [Bar89]. However, one must do it carefully, in a way that preserves the non-adaptivity of the decoding procedure. One option is to use Barrington's reduction and sub-exponential $\mathbf{AC}^0$ circuits for the semigroup problem (see [Tel20]). While this would give us an $\mathbf{AC}^0$ decoder, its size will be $k^{\Omega(1)}$ rather than $2^{\mathrm{polyloglog}(k)}$.

Employing either of the above would also allow us to bound the number of majority gates in our locally list-decodable code's decoder from Theorem 4.2 by $Q = \mathrm{poly}(\log k, 1/\varepsilon)$.

## B.3 A logspace computable lossless expander

**Claim B.2.** *Given positive integers $N$ and $K \leq N$, and any $\varepsilon > 0$ and a constant $\alpha > 0$, let $\Gamma\colon [N] \times [D] \to [M]$ be the $(K, \varepsilon)$ lossless expander of Theorem 5.4. Then, given $f \in [N]$ and $y \in [D]$, $\Gamma(x, y)$ can be computed in space $O(\log\log N + \log\log(1/\varepsilon))$.*

*Proof.* We first recall the construction and parameters of [KT22]. Denoting $n = \log N$, they set a prime field $\mathbb{F}_q$ for $q = \mathrm{poly}_\alpha(n/\varepsilon)$. Also, $m \in \mathbb{N}$ is such that $M = q^{m+2}$, and $[D] \equiv \mathbb{F}_q$. The parameters are set so that we can identify $f$ as a univariate polynomial over $\mathbb{F}_q$ with degree at most $n-1$. The output $\Gamma(f, y)$ is given by

$$\Gamma(f, y) = \left(y, f(y), f'(y), \ldots, f^{(m+1)}(y)\right) \in \mathbb{F}_q^{m+2}.$$

where $f^{(i)}$ is the (formal) $i$-th derivative of $f$. Fix some $i \in [m+1]$, and consider the computation of some $f^{(i)}(y)$. It is known that iterated addition and multiplication can be done by logspace-uniform (and even logtime uniform) $\mathbf{TC}^0$ circuits, so in particular in $\mathbf{L}$. More concretely, adding and multiplying $t$ $\mathbb{F}_q$-elements can be done by $\mathrm{poly}(t \log q)$-sized $\mathbf{TC}^0$ circuits (see, e.g., [HAB02]).[36] Computing $f^{(i)}(y)$ amounts to:

1. Computing $f^{(i)}$: Each coefficient of $f^{(i)}$ is a multiplication of $O(m)$ field elements, which can be computed in space $O(\log m + \log\log q)$.
2. Evaluating $f^{(i)}(y)$: Amounts to computing exponentiation up to an $n$-th power, and performing addition of up to $n$ elements. This can be done in space $O(\log n + \log\log q)$.

By composition of space bounded functions, Proposition 3.2, the overall space requirement of computing the $m$ field elements of $\Gamma(f, y)$ is

$$O(\log n + \log\log q + \log m) = O\left(\log\log N + \log\log\frac{1}{\varepsilon}\right). \quad \blacksquare$$

[36] Computing a field element to the $t$-th power can even be done in size $\mathrm{poly}(\log t, \log q)$ for specific realizations of $\mathbb{F}_q$, see [HV06], but we won't need this fact.



ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il