Electronic Colloquium on Computational Complexity, Report No. 80 (2021)

Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise

Lijie Chen [*] Roei Tell [†]

June 10, 2021

Abstract

We propose a new approach to the hardness-to-randomness framework and to the *promise-$\mathcal{BPP}$ = promise-$\mathcal{P}$* conjecture. Classical results rely on non-uniform hardness assumptions to construct derandomization algorithms that work in the worst-case, or rely on uniform hardness assumptions to construct derandomization algorithms that work only in the average-case. In both types of results, the derandomization algorithm is "black-box" and uses the standard PRG approach. In this work we present results that closely relate *new and natural uniform hardness assumptions* to *worst-case derandomization* of *promise-$\mathcal{BPP}$*, where the algorithms underlying the latter derandomization are *non-black-box*.

In our main result, we show that *promise-$\mathcal{BPP}$ = promise-$\mathcal{P}$* if the following holds: There exists a multi-output function computable by logspace-uniform circuits of polynomial size and depth n^2 that cannot be computed by uniform probabilistic algorithms in time n^c, for some universal constant $c > 1$, on *almost all inputs*. The required failure on "almost all inputs" is stronger than the standard requirement of failing on one input of each length; however, the same assumption without the depth restriction on f is *necessary* for the conclusion. This suggests a potential equivalence between worst-case derandomization of *promise-$\mathcal{BPP}$* of any form (i.e., not necessarily by a black-box algorithm) and the existence of efficiently-computable functions that are hard for probabilistic algorithms on almost all inputs.

In our second result, we introduce a new and uniform hardness-to-randomness tradeoff for the setting of *superfast average-case derandomization*; prior to this work, superfast average-case derandomization was known only under non-uniform hardness assumptions. In an extreme instantiation of our new tradeoff, under appealing uniform hardness assumptions, we show that for every polynomial $T(n)$ and constant $\epsilon > 0$ it holds that $\mathcal{BPTIME}[T] \subseteq$ heur-$\mathcal{DTIME}[T \cdot n^{\epsilon}]$, where the "heur" prefix means that no polynomial-time algorithm can find, with non-negligible probability, an input on which the deterministic simulation errs.

Technically, our approach is to design *targeted PRGs and HSGs*, as introduced by Goldreich (LNCS, 2011). The targeted PRGs/HSGs "produce randomness from the input", as suggested by Goldreich and Wigderson (RANDOM 2002); and their analysis relies on non-black-box versions of the reconstruction procedure of Impagliazzo and Wigderson (FOCS 1998). Our main reconstruction procedure crucially relies on the ideas underlying the proof system of Goldwasser, Kalai, and Rothblum (J. ACM 2015).

[*]Massachusetts Institute of Technology, Cambridge, MA. Email: `lijieche@mit.edu`
[†]Massachusetts Institute of Technology, Cambridge, MA. Email: `roei.tell@gmail.com`

ISSN 1433-8092

Contents

1 Introduction

One of the major achievements in complexity theory is the connection between pseudorandomness and lower bounds, which is known as the hardness-to-randomness framework. In a sequence of seminal works following [Yao82; BM84; Nis91; NW94; IW99], derandomization algorithms for *promise-$\mathcal{BPP}$* (denoted *pr$\mathcal{BPP}$*, in short) were constructed under the assumption that there is a function in $\mathcal{E} = \mathcal{DTIME}[2^{O(n)}]$ that is hard for *non-uniform circuits*. These derandomization algorithms work in a "black-box" fashion, by enumerating the seeds of a PRG and evaluating the relevant probabilistic algorithm on the resulting set. Since efficiently-computable PRGs (i.e., that run in time exponential in the seed length) are well-known to imply circuit lower bounds for $\mathcal{E}$, the bottom line of this line-of-works is that efficient black-box derandomization of $pr\mathcal{BPP}$ is essentially equivalent to circuit lower bounds for $\mathcal{E}$.

It is a classical open question whether or not the foregoing black-box approach is *necessary* for derandomization of $pr\mathcal{BPP}$. A natural suspicion is that this approach might be an "overkill", since the derandomization of a probabilistic machine M on input x does not depend on the way M operates on x (other than its number of steps), and just evaluates M on x using the output-set of a PRG as randomness. In the case of non-deterministic derandomization such as $pr\mathcal{BPP} \subseteq pr\mathcal{NP}$, we do know that this black-box approach is necessary (see, e.g., [IKW02; Wil13; MW18; Che19; CW19; CR20; CLW20]). However, for deterministic derandomization such as $pr\mathcal{BPP} = pr\mathcal{P}$ there has been essentially no unconditional progress on the question of whether black-box derandomization is necessary in the last three decades (see, e.g., [CRT+20; Tel19] for conditional results and further discussion).

In this work we show that *the hardness-to-randomness framework can be revised* into a form that completely avoids the foregoing question. We prove a new and general hardness-to-randomness tradeoff that closely relates *uniform hardness assumptions*, of a particular type that we introduce, to *non-black-box derandomization of $pr\mathcal{BPP}$*. That is, under the foregoing uniform hardness assumptions we show how to deduce strong derandomization conclusions such as $pr\mathcal{BPP} = pr\mathcal{P}$, and we complement this result by showing that similar hardness assumptions are *necessary* for the derandomization conclusion. Thus, our approach suggests an appealing path towards proving an *equivalence* between *any derandomization* of $pr\mathcal{BPP}$ (i.e., not necessarily a black-box derandomization) and corresponding *uniform lower bounds* of the foregoing type.

In addition, mirroring classical hardness-to-randomness results, the new approach is general enough to allow trading off the hypothesis for the conclusion both in terms of running time and in terms of other structural restrictions on the probabilistic algorithms (e.g., we show hardness-to-randomness tradeoffs for algorithms that work in parallel). In several settings that are obtained by such "scaling", our results already come close to proving a full equivalence between the hypothesized uniform lower bound and the derandomization conclusion.

The general form of our hardness hypotheses. Generally speaking, the uniform hardness hypotheses that we will use towards derandomization of $pr\mathcal{BPP}$ are of the following form:

> There exists a multi-output function $f\colon \{0,1\}^n \to \{0,1\}^{k(n)}$ that can be computed by a deterministic algorithm that runs in time $T(n)^{O(1)}$ and satisfies an additional efficiency requirement, but cannot be computed by probabilistic algorithms in time $T(n)$.

The precise technical meanings of "additional efficiency requirement" and of "cannot be computed" vary across our particular results below. However, we stress that this form of hypothesis *does not assume* that the function f is hard for non-uniform circuits, or that the string $f(x)$ (for some input $x \in \{0,1\}^n$) is a truth-table of a function that is hard for non-uniform circuits. What we assume is simply that *given x, it is hard to print the string $f(x)$* in probabilistic time $T(|x|)$. This hardness assumption does not refer to circuit complexity at all.[1]

Our work in context: Uniform hardness-to-randomness. Numerous previous works deduced *some* derandomization conclusion from assumptions asserting that an efficiently-computable function is hard for uniform probabilistic algorithms.[2] However, *the conclusions that we deduce from the assumptions in this work are considerably stronger*. In more detail, the line-of-works following Impagliazzo and Wigderson [IW98] uses standard uniform hardness assumptions (such as $\mathcal{BPP} \neq \mathcal{EXP}$) to construct PRGs that derandomize $\mathcal{BPP}$ in the *average-case*. (In fact, these works typically show that their assumptions are equivalent to average-case derandomization.) Besides working only "on average", the derandomization typically also works only on infinitely-many input lengths, and is relatively slow, even when the hypothesized lower bound is strong (see [CIS18; CRT+20] for recent results and for discussions of previous results).

We use our new uniform hardness assumptions to construct derandomization algorithms that work in the *worst-case, on almost all input lengths, and can work in polynomial time* (or, in another result, to construct an extremely fast average-case derandomization that was not previously known to follow from uniform assumptions). Thus, our results strike a better balance than previous hardness-to-randomness tradeoffs: In contrast to the average-case results above, we deduce strong conclusions such as $pr\mathcal{BPP} = pr\mathcal{P}$, which demonstrates that *our assumptions are not "too weak"*; and in contrast to results that rely on non-uniform hardness assumptions, we prove that uniform hardness assumptions similar to the ones that we use are *necessary for derandomization*, suggesting that our hardness assumptions might not be "too strong". Indeed, the reason that we are able to strike this better balance is since our derandomization algorithms work in a non-black-box fashion. See Figure 1 for a visual comparison with known results.

1.1 Derandomization from hardness on almost all inputs

Our first main result is motivated by the following observation. Recall that if $pr\mathcal{BPP} = pr\mathcal{P}$, then for every $c > 1$ there exists $f \in \mathcal{P}$ that is hard for $\mathcal{BPTIME}[n^c]$ on almost all input lengths. (This is because we can derandomize $\mathcal{BPTIME}[n^c] \subseteq \mathcal{DTIME}[n^{O(c)}]$ and then appeal to a standard time-hierarchy theorem). The observation is that if we take f to have multiple outputs, then we can deduce that every probabilistic n^c-time algorithm fails to compute f *on almost all inputs*, rather than only on some input for every large enough input length; that is, for every probabilistic n^c-time algorithm there exists $n_0 \in \mathbb{N}$ such that the probabilistic algorithm fails to compute f on *each and every input $x \in \{0,1\}^*$ of length $|x| \geq n_0$*.

[1] Our notion of hardness is reminiscent of a lower bound on the conditional randomized time-bounded Kolmogorov complexity $\mathtt{rK}^T(f(x)|x)$ of $f(x)$; see [Oli19; LOS21; OL21; LP21] for recent studies of closely related notions.

[2] We note that hardness for *non-deterministic* probabilistic algorithms (i.e., for $\mathcal{MA}$) is a significantly stronger assumption, which is in fact sufficiently strong to imply the non-uniform hardness assumptions needed for worst-case derandomization using PRGs (this follows from known Karp-Lipton-style theorems such as [BFN+93]). We thus limit our comparison to results that use hardness assumptions for standard probabilistic algorithms.

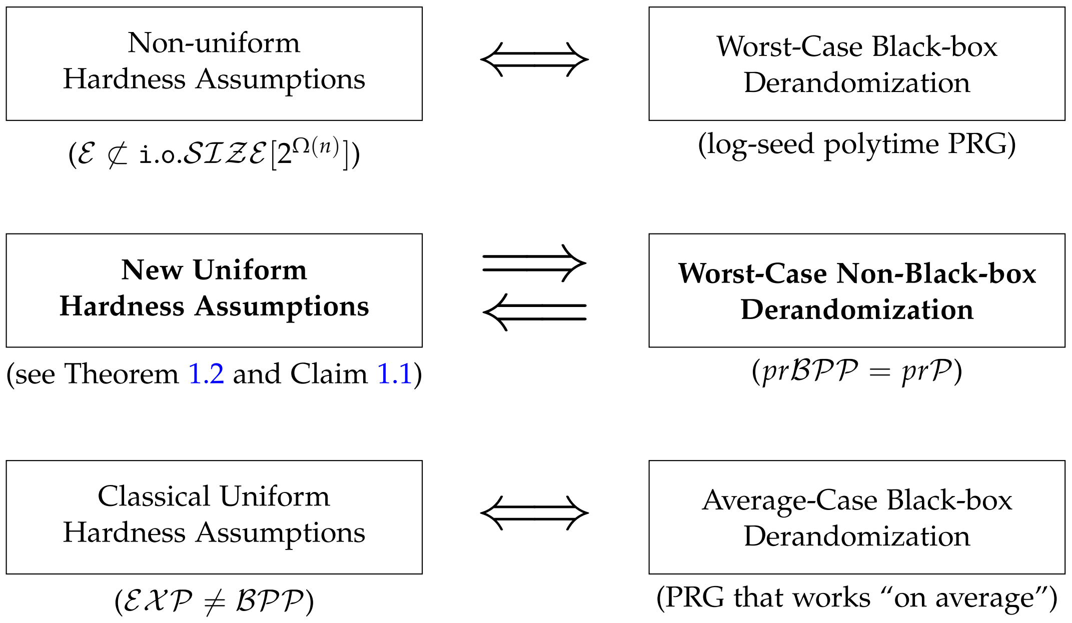

Non-uniform Hardness Assumptions
$\Longleftrightarrow$
Worst-Case Black-box Derandomization
$(\mathcal{E} \not\subset \texttt{i.o.}\mathcal{SIZE}[2^{\Omega(n)}])$
(log-seed polytime PRG)
New Uniform Hardness Assumptions
$\Longrightarrow$
$\Longleftarrow$
Worst-Case Non-Black-box Derandomization
(see Theorem 1.2 and Claim 1.1)
$(pr\mathcal{BPP} = pr\mathcal{P})$
Classical Uniform Hardness Assumptions
$\Longleftrightarrow$
Average-Case Black-box Derandomization
$(\mathcal{EXP} \neq \mathcal{BPP})$
(PRG that works "on average")

Figure 1: Classical hardness-to-randomness results compared with our new results for worst-case derandomization. For concreteness, below each box we mention in parentheses a specific parameter setting of the general result. The top row was proved by [IW99] (a smooth parametric scaling was subsequently shown in [Uma03]), the bottom row was proved by [IW98] (non-smooth parametric scalings were shown in subsequent works such as [TV07; CRT+20]), and the middle row corresponds to the results in Section 1.1.

Claim 1.1 (almost-all-inputs hardness is necessary for derandomization)**.** *If $pr\mathcal{BPP} = pr\mathcal{P}$, then for every $c \in \mathbb{N}$ there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in deterministic polynomial time such that f cannot be computed in probabilistic time n^c on* almost all inputs.

The proof of Claim 1.1 amounts to diagonalizing against every probabilistic machine that runs in time n^c on almost every input, by using each output bit of f to diagonalize against a different machine (and relying on the derandomization hypothesis); see Claim 5.2 for details.[3]

Our main result is that the existence of f as in conclusion of Claim 1.1 in fact *suffices* to deduce that $pr\mathcal{BPP} = pr\mathcal{P}$, assuming one additional structural restriction. Specifically, to deduce that $pr\mathcal{BPP} = pr\mathcal{P}$ we need the "almost-all-inputs" hard function f to be computable not just in $\mathcal{P}$, but also by uniform circuits of depth that is noticeably smaller than their size. For example, it suffices to assume that f is computable by circuits of depth n^2:

Theorem 1.2 (polynomial-time non-black-box derandomization of $pr\mathcal{BPP}$)**.** *Assume that there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform circuits of polynomial size and depth n^2 such that f cannot be computed in probabilistic time n^c on* almost all inputs, *where $c > 1$ is a sufficiently large universal constant. Then $pr\mathcal{BPP} = pr\mathcal{P}$.*

We can replace the depth n^2 in the hypothesis of Theorem 1.2 by an arbitrary fixed depth n^k, at the expense of increasing the constant $c = c_k$ (setting $c > k + O(1)$ suffices; see Corollary 5.7).

[3]Indeed, it is not necessary for f to have precisely n output bits, and our main result also does not depend on f having precisely n output bits. We chose this output length for simplicity of presentation.

We stress that the notion of uniformity in Theorem 1.2 (i.e., logspace-uniformity) is meaningful only due to the depth constraint,[4] and is considerably weaker than standard notions of uniformity for circuits (such as DLOGTIME-uniformity). Thus, the main gap between the hypothesis that suffices to deduce that $pr\mathcal{BPP} = pr\mathcal{P}$ and the hypothesis that is necessary for this conclusion is that in the former the "almost-all-inputs" hard function can be computed in parallel (e.g., by $\mathrm{poly}(n)$ processors running in parallel in time n^2).

1.1.1 Extensions: Scaling the new hardness-to-randomness result

Recall that classical hardness-to-randomness results extend both to derandomization of restricted complexity classes (such as $pr\mathcal{BP}\cdot\mathcal{NC}$), and to "low-end" results, which are tradeoffs between weaker lower bounds and slower derandomization. We now state several extensions of Theorem 1.2, demonstrating that:

1. The hardness-to-randomness result in Theorem 1.2 *scales quite well*, both to restricted circuit classes and to weaker parameter settings.
2. When scaling the result in *either* of the foregoing two directions, the gap between the required lower bound and the necessary one *considerably narrows*.

As a first extension, we show that quantitatively weaker "almost-all-inputs" lower bounds imply slower derandomization. The tradeoff we obtain is very smooth when deducing derandomization of $pr\mathcal{RP}$, and less smooth when deducing derandomization of $pr\mathcal{BPP}$; specifically, the following result is a more general version of Theorem 1.2:

Theorem 1.3 (non-black-box derandomization of $pr\mathcal{BPP}$; a general version of Theorem 1.2)**.** *There exists a universal constant $c > 1$ such that the following holds. Assume that there exists a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform circuits of size $T(n)$ and depth $d(n)$ that cannot be computed in probabilistic time $d(n)\cdot n^c$ on* almost all inputs. *Then, we have that*

$$pr\mathcal{RP} \subseteq \bigcup_{a\geq 1} pr\mathcal{DTIME}[\bar{T}(n^a)]$$
$$pr\mathcal{BPP} \subseteq \bigcup_{a\geq 1} pr\mathcal{DTIME}[\bar{T}(\bar{T}(n^{c\cdot a})^a)]\,,$$

where $\bar{T}(m) = 2^{c\cdot\log^2(T(m))/\log(m)}$.

To see that Theorem 1.3 is a more general version of Theorem 1.2, note that when T is a polynomial then $\bar{T}(n^a)$ and $\bar{T}(\bar{T}(n^{c\cdot a})^a)$ are also polynomials. Similarly, as another example, note that when T is quasi-polynomial then $\bar{T}(n^a)$ and $\bar{T}(\bar{T}(n^{c\cdot a})^a)$ are also quasi-polynomials. A natural instantiation of Theorem 1.3 is with $d(n) = \mathrm{poly}(n)$, in which case the hypothesized hardness is for probabilistic polynomial-time algorithms, but other instantiations are also useful; see Section 1.1.2 for details.

When scaling our tradeoff even further to an extreme "low-end" setting, we are able to *completely eliminate the structural restrictions* on f. Specifically, in order to derandomize $pr\mathcal{RP}$

[4]This is since the classical transformations of uniform Turing machines to uniform circuits yield circuits that are logspace-uniform (and in fact even "more uniform", e.g. DPOLYLOGTIME-uniform).

infinitely-often in fixed exponential-time 2^{n^c} for some $c \geq 1$ (indeed a very weak derandomization), we do not need to assume that the hard function is computable by uniform circuits of bounded depth.

Theorem 1.4 (non-black-box fixed-exponential-time derandomization of $\mathcal{RP}$)**.** *Assume that there exists a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in time 2^{n^c}, where $c \geq 1$ is a constant, such that f cannot be computed by probabilistic polynomial-time algorithms infinitely-often on almost all inputs.*[5] *Then, for some constant $c' \geq 1$ it holds that $pr\mathcal{RP} \subseteq \texttt{i.o.}\mathcal{DTIME}[2^{n^{c'}}]$.*

Indeed, Theorem 1.4 already comes close to a full equivalence between the hardness hypothesis and the derandomization conclusion; essentially, the only remaining gap is that we derandomize $pr\mathcal{RP}$ rather than $pr\mathcal{BPP}$. See Theorem 5.17 for details.

We suspect that it is possible to improve the derandomization overhead in Theorem 1.3 (see Remark 5.5), but we did not optimize our construction towards this purpose. Instead, we optimized our construction to scale down to *weak circuit classes*, in which case both the hardness hypothesis and the derandomization conclusion scale down to the relevant circuit class. The weakest natural class for which the result holds is that of logspace-uniform $\mathcal{NC}$ circuits; that is, logspace-uniform circuits of polynomial size and polylogarithmic depth. Towards stating the results, recall that $\mathcal{NC}^i$ is the class of polynomial-sized circuits of depth $\log^i(n)$; then:

Theorem 1.5 (non-black-box derandomization of $\mathcal{NC}$)**.** *There exists a universal constant $c > 1$ such that the following holds. Assume that for every sufficiently large $i \in \mathbb{N}$ there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in logspace-uniform deterministic $\mathcal{NC}^{(.99 \cdot i)}$ that cannot be computed by logspace-uniform probabilistic $\mathcal{NC}^i[n^c]$ on* almost all inputs. *Then,* logspace-uniform-$pr\mathcal{BP} \cdot \mathcal{NC} \subseteq$ logspace-uniform-$pr\mathcal{NC}$.

Note that, similarly to the "low-end" extensions above, the structural requirements on the hard function in Theorem 1.5 are relaxed. First, the logspace-uniformity of circuits for f is now called for, since we want the derandomization algorithm to be computable by such circuits. And secondly, the gap in depth (between the deterministic circuits for f and the probabilistic circuits for which f is hard) is now only polylogarithmic. See Corollary 5.15 for further details.

1.1.2 Robustness of our results: Relaxing the hypotheses

One may ask what happens if the hardness in (say) Theorem 1.3 holds not with respect to almost all inputs, but "only" with respect to some very dense set of inputs. In all of the results in this section, when one assumes hardness with respect to an (arbitrary) distribution $\mathbf{x}$ and some succeess bound $\mu > 0$, we can deduce derandomization of probabilistic algorithms that have one-sided error with respect to the precise same distribution $\mathbf{x}$ and with μ as the error parameter. (This is because our hardness-to-randomness tradeoffs are "instance-wise", and hold with respect to any fixed input x; we explain this in Section 1.3.)

Indeed, the tradeoff emphasized above between almost-all-inputs hardness and worst-case derandomization simply represents the extreme setting when $\mathbf{x}$ is supported on all inputs and

[5]The meaning of "infinitely-often hard on almost all inputs" is that there is an infinite set $S \subseteq \mathbb{N}$ of input lengths such that for every probabilistic algorithm and every sufficiently large $n \in S$ and every $x \in \{0,1\}^n$, the algorithm fails to compute f on x.

$\mu = 0$ (in which case we reduce derandomization of probabilistic algorithms that have two-sided error to derandomization of probabilistic algorithms that have one-sided error).

Another type of tradeoff, which can be obtained when considering derandomization in super-polynomial time, allows to almost completely eliminate the depth constraint on f. Specifically, to deduce worst-case derandomization of $pr\mathcal{BPP}$ in (say) quasipolynomial time, it suffices to assume that for some constant $\epsilon > 0$, the function f is computable by deterministic logspace-uniform circuits of quasipolynomial size T and of depth $T(n)^{1-\epsilon}$, but cannot be computed in probabilistic time $T(n)^{1-\epsilon/4}$ on almost all inputs (see Corollary 5.8 for precise details). In comparison to Theorem 1.2, this lower bound is stricter in terms of time (intuitively, we need a function in time T that is hard for time $T^{.99}$), but poses almost no depth restriction on f.

1.1.3 Discussion: Derandomization vs "almost-all-inputs" lower bounds

The main implication of the results in this section is a new and tight connection between **worst-case derandomization** and **almost-all-inputs lower bounds** for probabilistic algorithms, which is the connection hinted at in Figure 1. Our results demonstrate that this is a *general* connection, which manifests itself in different parametric settings as well as for classes of restricted algorithms, such as parallel algorithms. (Some technical elaborations on the two latter connections, beyond what was already stated above, appear in Theorem 5.17 and Corollary 5.15.)

We stress that this new connection is *interesting per-se*, and not only since it improves on classical hardness-to-randomness results (i.e., not only since it manages to get worst-case derandomization without relying on PRGs and on the corresponding non-uniform lower bounds). First, this connection highlights the importance of a natural problem in theoretical computer science that, as far as we are aware, received little attention so far (i.e., almost-all-inputs lower bounds). Secondly, the connection sheds additional light on the central role of derandomization in complexity theory, by significantly refining its connections to questions of lower bounds.

And thirdly, this connection is a very natural one. Loosely speaking, it relates the statement "deterministic machines can efficiently compute a *hard function* for probabilistic machines" to the statement "deterministic machines can efficiently *simulate* probabilistic machines"; that is, we relate simulation to lower bounds for these classes of machines. This can be viewed as demonstrating a setting with a partial affirmative answer to a classical question asked by Kozen [Koz78]: Informally, he asked for which pairs of complexity classes is simulation equivalent to lower bounds (pointing out the implication that in this case diagonalization is a complete method for lower bounds).[6] The interesting and non-standard part in the answer suggested by our results is that the hardness refers to multi-output functions and to almost-all-inputs hardness.

1.2 Superfast derandomization from non-batch-computable functions

Our second main result refers to *superfast derandomization* of $\mathcal{BPP}$, meaning derandomization that has almost no time overhead. The question of the best possible polynomial time overhead was recently raised by Doron *et al.* [DMO+20], who showed a conditional derandomization with near-quadratic overhead. In a follow-up work, the current authors showed [CT21] (among other

[6]Needless to say, this is a very informal phrasing of his broad question, which can be interpreted in various different ways to begin with. For further study and some formalizations see [Koz78; NIR03; NIR06].

results) a conditional average-case derandomization with *almost no overhead*: For every $\epsilon > 0$, probabilistic algorithms that run in time T were simulated by deterministic algorithms that run in time $T \cdot n^{\epsilon}$, on average-case over any T-time samplable distribution.

The hardness hypotheses in both previous works were very strong non-uniform lower bounds.[7] Our second main result is a new hardness-to-randomness approach for the setting of superfast derandomization that relies only on *appealing uniform hardness hypotheses*. As in Section 1.1, we will consider a multi-output function $f\colon \{0,1\}^n \to \{0,1\}^{k(n)}$ that is computable in deterministic time T but not in smaller probabilistic time T'; and since we now consider superfast derandomization, the gap between T and T' will now be very small. However, in the current setting our structural requirement on f will be different: We will assume that *each individual output bit of $f(x)$* can be computed faster than the time-complexity of *printing the entire string $f(x)$*.

Intuitively, we refer to such hard functions as non-batch-computable: This is since the hardness assumption implies that when printing all the bits of the string $f(x)$ in a batch, one has to invest noticeably larger running time than when printing just a single bit of $f(x)$.

1.2.1 The most appealing special case: A direct-product hypothesis

We first describe what we consider to be the most appealing special case of our result. Recall that standard *direct-product results* start with a function f_0 that is hard to compute in time T on (say) $1/3$ of the inputs, and prove that its k-wise direct product $f = f_0^{\times k}$ is hard to compute in time $T' \approx T$ on $1 - 2^{-\Omega(k)}$ of the inputs (see, e.g., [IJK+10]).[8] Our starting point is that these results are also known to extend (unconditionally) to "approximate-direct-product" versions, showing that $f(z)$ cannot even be *approximately-printed* for the vast majority of z's: That is, every time-T' algorithm fails, for .99 of the tuples $z = (x_1, ..., x_k)$, to even output a string $\widetilde{f}(z)$ such that $\Pr_{i\in[k]}\left[\widetilde{f}(z)_i = f(z)_i\right] \geq .99$. See Proposition 6.10 for a precise statement, following [IJK+10].

A well-known conjecture is that in some cases f is harder also for algorithms with larger running time; taken to the extreme, the "strong direct-product" conjecture asserts that in some cases one cannot improve on the naive algorithm that runs in time $k \cdot T$ (see, e.g., [Sha03]). Our hypothesis will be that a mildly-strong approximate-direct-product result holds for *some function* $f_0 \in \mathcal{DTIME}[T]$ that is hard for time $T \cdot n^{-\Omega(1)}$. Specifically, given a parameter $\delta > 0$, we assume that for every $T(n) = \text{poly}(n)$ and $k(n) = n^{\Omega(1)}$ there exists $f_0 \in \mathcal{DTIME}[T]$ such that $f = f_0^{\times k}$ satisfies the following: Every probabilistic algorithm that gets input $z \in \{0,1\}^{n \cdot k(n)}$ and runs in time $T(n) \cdot k(n)^{\alpha}$ does not succeed in printing a string $\widetilde{f}(z)$ satisfying $\Pr_{i\in[k(n)]}\left[\widetilde{f}(z)_i = f(z)_i\right] \geq 1 - \alpha$ on more than a δ-fraction of z's, where $\alpha > 0$ can be an arbitrarily small constant (see Assumption 6.11 for a formal statement).

As in our previous work, to deduce derandomization with essentially no time overhead we will need another assumption, albeit a completely standard one – the existence of one-way functions. Assuming that the mildly-strong approximate-direct-product hypothesis holds, and that one-way functions secure against uniform polynomial-time algorithms exist, we show that $\mathcal{BPP}$

[7]Specifically, the assumption in [DMO+20] was that there exists a problem in $\mathcal{DTIME}[2^n]$ that is hard for randomized non-deterministic circuits of size $\approx 2^{.99n}$. The assumptions in [CT21] were of that non-uniformly secure one-way functions exist, and that there exists a problem in time $2^{k \cdot n}$ that cannot be solved in time $2^{.99k \cdot n}$ even with $2^{.99n}$ bits of non-uniform advice (where $k = k_{T,\epsilon}$ is a suitable large constant).

[8]Recall that $f = f_0^k$ is defined by $f(x_1, ..., x_k) = (f_0(x_1), ..., f_0(x_k))$.

can be simulated, on average, with almost no time overhead:

Theorem 1.6 (superfast non-black-box derandomization from mildly-strong approximate-direct-product)**.** *Assume that one-way functions exist, and that for some $\delta > 0$ the mildly-strong approximate-direct-product hypothesis holds. Then, for every polynomial T we have that*[9]

$$\mathcal{BPTIME}[T] \subseteq \bigcap_{\epsilon>0} \mathsf{avg}_{1-2\delta}\text{-}\mathcal{DTIME}[n^{\epsilon} \cdot T] \ .$$

We view the hypothesis in Theorem 1.6 as quite mild considering the strong conclusion. First, as mentioned above, the existence of $f_0 \in \mathcal{DTIME}[T] \setminus \mathcal{BPTIME}[T \cdot n^{-\Omega(1)}]$ is necessary for superfast derandomization. Secondly, the mildly-strong approximate-direct-product hypothesis is a relaxed one: We only assume the existence of *one* suitable f_0 (rather than a direct-product result for a class of functions), and only assume that the hardness grows to $T \cdot k^{.0001}$ (rather than to $T \cdot k$). And thirdly, in contrast to [CT21], the hypothesis in Theorem 1.6 is completely uniform, since even the one-way function only needs to be uniformly-secure (in [CT21] we needed a one-way function secure against non-uniform circuits).

1.2.2 Randomness might be "indistinguishable from useless" for decision problems

The drawback of the concluded derandomization in Theorem 1.6 is that it succeeds only over the uniform distribution. Recall that in our previous work [CT21], assuming strong non-uniform lower bounds, the derandomization succeeded over all T-time samplable distributions.

In the following result we show that a natural strengthening of the uniform hypothesis in Theorem 1.6 allows to derandomize $\mathcal{BPP}$ with the same tiny time overhead over *all polynomial-time-samplable distributions* and with a *negligible average-case error*. Informally, an appealing interpretation of the latter conclusion is that randomness is "indistinguishable from being essentially useless" for solving decision problems in polynomial time (where "essentially useless" here means that it can save a factor of at most n^{ϵ} in the running time).

Towards stating the result, let us say that a probabilistic algorithm A approximately prints a function $f \colon \{0,1\}^n \to \{0,1\}^{k(n)}$ on input x if $\Pr[A(x)_i = f(x)_i] \geq .99$, where the probability is over $i \in [k(n)]$ and the internal randomness of A (note that the input x is fixed). Then:

Theorem 1.7 (superfast non-black-box derandomization over all polynomial-time-samplable distributions; see Theorem 6.5)**.** *Let $T \colon \mathbb{N} \to \mathbb{N}$ be a polynomial. Assume that one-way functions exist, and that for every $\epsilon > 0$ there exist $\delta > 0$ and a function $f \colon \{0,1\}^n \to \{0,1\}^{n^{\epsilon}}$ such that:*

1. *There exists an algorithm that gets input $(x,i) \in \{0,1\}^n \times [n^{\epsilon}]$ and outputs the i^{th} bit of $f(x)$ in time $T'(n)$, where $T'(n) = T(n) \cdot n^{\epsilon}$.*
2. *For every probabilistic algorithm A that runs in time $T'(n) \cdot n^{\delta}$ and every polynomial-time samplable distribution $\mathbf{x}$, the probability over $x \sim \mathbf{x}$ that A approximately prints f on input x is negligible.*

Then $\mathcal{BPTIME}[T] \subseteq \bigcap_{\epsilon>0} \mathsf{heur}_{1-\mathsf{neg}}\text{-}\mathcal{DTIME}[n^{\epsilon} \cdot T]$, where neg is a negligible function.[10]

[9] The notation $\mathsf{avg}_{1-2\delta}$- means that for every L there exists a deterministic algorithm that errs with probability at most 2δ over a uniformly-chosen input.

[10] The notation $\mathsf{heur}_{1-\mathsf{neg}}$- means that for every L there exists a deterministic algorithm A_L such that for every polynomial-time-samplable distribution $\mathbf{x}$, the probability over $x \sim \mathbf{x}$ that A_L errs on x is negligible.

We note that the conclusion of Theorem 1.7 can *only be obtained using a non-black-box derandomization algorithm* (such as the one that we use); that is, any PRG-based approach cannot yield such a conclusion. See Remark 6.6 for an explanation.

As we explain below (see Section 1.3), all of our derandomization algorithms not only solve the decision problem, but also *find* random strings that lead the relevant probabilistic machine to a correct decision. This has the following implication to the question of *explicit constructions*.

We say that an algorithm A is an explicit construction of objects from a set $\Pi \subseteq \{0,1\}^*$ if $A(1^n)$ prints an n-bit string in Π. We are particularly interested in deterministic explicit constructions of random-looking objects, which corresponds to the case where Π is dense (e.g., $|\Pi \cap \{0,1\}^n| \geq 2^n/2$). In this case, we show that under the same assumption as in Theorem 1.7, *the complexity of explicit constructions is nearly identical to that of verifying that the object has the specified property*; that is, loosely speaking, deterministically constructing good random-looking objects is never significantly more expensive than deciding if an object is good:

Corollary 1.8 (superfast explicit constructions)**.** *Suppose that the assumption of Theorem 1.7 holds for every polynomial T. Then, for every $\Pi \in \mathcal{BPTIME}[n^k]$ such that $|\Pi \cap \{0,1\}^n| \geq 2^n/n^{o(1)}$ and every $\epsilon > 0$, there exists an explicit construction of objects from Π in deterministic time $n^{k+\epsilon}$.*

Lastly, our results also allow to interpolate between the derandomization over the uniform distribution as in Theorem 1.6 and the derandomization over all polynomial-time-samplable distributions as in Theorem 1.7. Specifically, if a hypothesis similar to the one in Theorem 1.7 holds with respect to some fixed polynomial-time-samplable distribution $\mathbf{x}$, then the derandomization conclusion holds over this distribution $\mathbf{x}$ (see Theorem 6.4 for details).

1.2.3 Necessity of non-batch-computable functions for superfast derandomization

We do not know if "non-batch-computable" functions as in the hypotheses of Theorems 1.6 and 1.7 are necessary for average-case superfast derandomization *in general*. However, we prove that a non-batch-computable function is necessary in one *natural special case*.

Specifically, consider balanced formulas of polynomial size that are DLOGTIME-uniform, which we refer to as well-structured formulas. Loosely speaking, we show that if probabilistic well-structured formulas can be derandomized with overhead $T \mapsto T \cdot n^{\epsilon}$ on average over the uniform distribution, then the following holds: There exists a function $g\colon \{0,1\}^n \to \{0,1\}^{n^{\delta}}$ such that the mapping $(x,i) \mapsto g(x)_i$ is computable in time $\tilde{O}(T)$, but every probabilistic well-structured formula of size $o(T \cdot n^{\delta-\epsilon})$ fails to print $g(x)$, with high probability over uniform choice of x. For precise details (and a slightly stronger statement) see Proposition 6.15.

1.3 Non-black-box derandomization and instance-wise hardness

As mentioned above, our derandomization algorithms for all results above will not rely on the standard approach of enumerating the seeds of a PRG. Instead, our algorithms rely on what was defined by Goldreich [Gol11a] as targeted PRGs: These are PRGs that do not "fool" all circuits, but rather get an input x and fool all efficient uniform machines that also have access to x.[11]

[11]In [Gol11a] and in a subsequent work [Gol11b] Goldreich proposed two definitions for targeted PRGs, which he called targeted canonical derandomizers (and targeted canonical hitters for the HSG version). We use the first of those definitions, which appears in [Gol11a, Definition 4.1].

In other words, the potential distinguisher for the PRG is not modeled as a non-uniform circuit (as in the classical "textbook" approach, where the non-uniformity represents any possible fixed input), but rather by a uniform Turing machine that only gets access to the *particular fixed input* x that is also given to the PRG. Indeed, such targeted PRGs suffice for derandomization, and Goldreich in fact showed that their existence is also necessary for derandomization.

Recall that a complete problem for $pr\mathcal{BPP}$ is the Circuit Acceptance Probability Problem (CAPP): In this problem we are given a circuit and want to approximate its acceptance probability, up to an additive error of (say) $1/10$. The classical PRG approach for solving this problem treats the given circuit as a black-box, instantiating the PRG using only the *circuit size* as information. In contrast, our targeted PRGs *treat the given circuit in a non-black-box way*, using the description of the circuit as information in order to generate pseudorandom strings that are good for this particular circuit; see further details in Section 2.

Following classical techniques, our technical approach is *reconstructive*: We design a "reconstruction" algorithm R that transforms every distinguisher for the targeted PRG to an efficient procedure that computes the hard function. The innovative aspect in our reconstruction arguments is that they work instance-wise, for *each fixed input* x: That is, our targeted PRG maps every x to a collection S_x of pseudorandom strings, and our reconstruction algorithm R gets input x and access to a distinguisher for S_x, and prints the output of the hard function f at x (i.e., it prints $f(x)$). Thus, the hardness guarantee for a particular fixed x implies pseudorandomness of S_x for distinguishers that get access to that particular fixed x; in other words, *our hardness-to-randomness tradeoff holds with respect to each particular input* x. Accordingly, when we assume almost-all-inputs hardness of f we get worst-case derandomization as in Section 1.1, and when we assume average-case hardness of f we get average-case derandomization as in Section 1.2.

1.4 Organization

In Section 2 we describe our proof techniques, in high level. Section 3 includes preliminary definitions and statements of well-known results. In Section 4 we present the technical results underlying the proofs of the results that were described in Section 1.1, and we prove the foregoing results in Section 5. Finally, in Section 6 we prove the results that were described in Section 1.2. The appendices include proofs of several technical results that are stated and used in the paper.

2 Technical overview

We first explain the ideas behind our new hardness-to-randomness approach, in very high-level, and relate them to previous works and known approaches. Then, in Sections 2.1 and 2.2 we describe the proofs of our main results in more detail.

As mentioned in Section 1.3, the underlying setting is the following: We are given an input $x \in \{0,1\}^n$ and want to produce a collection of strings S_x that appear random to any distinguisher that is also given the same input x. At a bird's eye, our proof techniques merge the techniques from the two main lines-of-work concerning uniform hardness-to-randomness, which were separate so far: The line-of-works following Impagliazzo and Wigderson [IW98] (see, e.g., [CNS99;

Kab01; Lu01; GSTS03; TV07; SU07; GV08; CIS18; CRT+20]) and the line-of-works following Goldreich and Wigderson [GW02] (see, e.g., [MS05; Zim08; Sha11; KMS12; SW13; Alm19; Hoz19]).

The first main idea (extending [GW02]): Applying a hard function to the input. As a starting point, let us return to the original idea of Goldreich and Wigderson [GW02] – using the *information in the input* to produce good pseudorandom strings. In their original work they applied a randomness extractor Ext to the input x to obtain a list $\{\texttt{Ext}(x,s)\}_s$ of pseudorandom strings. A later modification of their approach by Kinne, van Melkebeek and Shaltiel [KMS12] applied a seed-extending pseudorandom generator G to obtain a pseudorandom string $G(x)$, using the input as a seed (see [KMS12] for the definition of a seed-extending PRG).

These two approaches naturally give rise to the following general question: Which function f can we apply to the input x in order to obtain good pseudorandom strings? The answer presented in the current work seems, in retrospect, to be the most straightforward one: *We apply a hard function f to the input x*, as the first step in our constructions of S_x.

As observed in [GW02], the main danger in using the input x as a source of randomness is the correlation between x and the set of random strings that lead the relevant probabilistic algorithm to a wrong decision on x. A key insight implicit in [Sha11; KMS12] is that this correlation is computational (i.e., can be recognized by an efficient algorithm), and thus we can avoid this danger by enforcing a suitable hardness requirement on S_x. Our use of a computationally-hard function $x \mapsto f(x)$ as the first step in constructing S_x can be viewed as following this idea.

The second main idea (extending [IW98]): A non-black-box uniform reconstruction argument. As mentioned in Section 1.3, we do not just output $f(x)$, but show efficient transformations of $f(x)$ to a set $S_x = S_{f,x}$ of pseudorandom strings such that any distinguisher D for S_x can be efficiently transformed to an algorithm F satisfying $F(x) = f(x)$. The mapping of x to $f(x)$ and then to the set S_x is the core of our targeted PRGs, and the transformation of a distinguisher for S_x to an algorithm F is called an *instance-wise reconstruction procedure*.

The main previously-known approach for designing uniform reconstruction procedures, introduced by [IW98; TV07], requires that the hard function will be downward self-reducible and randomly self-reducible, and only allows to deduce that the PRG works on average-case and infinitely-often (we explain this in Section 2.2). We show a "non-black-box" version of this argument that crucially relies on access to the input x, and that works whenever the hard function is computable by *logspace-uniform circuits of bounded depth* (e.g., of size $T(n)$ and depth $\sqrt{T(n)}$). Indeed, this new version allows to deduce that the derandomization works in the worst-case and on almost all input lengths. Our reconstruction procedure is based on the ideas in the doubly-efficient proof system of Goldwasser, Kalai, and Rothblum [GKR15].

2.1 Warm-up: Proofs of Theorems 1.6 and 1.7

Let us begin by proving Theorems 1.6 and 1.7. The proofs of these results are simpler, and showcase our ideas of applying a hard function to the input and of analyzing a targeted PRG by "instance-wise" reconstruction. Our goal now will be to derandomize probabilistic algorithms running in time $T(n) = \text{poly}(n)$ in deterministic time $n^{\epsilon} \cdot T(n)$ on average, and for simplicity we consider derandomization over the uniform distribution.

The main idea. Let us first explain our idea while ignoring the precise parameters. We think of $f(x)$ as a hard truth-table that is produced from the input x, and use this truth-table to instantiate a suitable version of the reconstructive PRG of [NW94; IW99] (see below). As observed in [IW98], the reconstruction procedure for this PRG is a *uniform learning algorithm* that works when given access to a distinguisher for the PRG; that is, the reconstruction procedure gets access to a distinguisher D, issues a small number of queries to the function described by $f(x)$,[12] and outputs a small oracle circuit C such that C^D computes the function described by $f(x)$.

The main pressing question is how our reconstruction procedure can answer the queries of this learning algorithm without using any non-uniformity. This is where our assumption that the function $(x,i) \mapsto f(x)_i$ is efficiently-computable comes in. Specifically, assume that we can compute $(x,i) \mapsto f(x)_i$ in time $T'(n) = \text{poly}(n)$, but that printing all of $f(x)$ cannot be done in time much smaller than $|f(x)| \cdot T'(n)$. Then, our reconstruction argument relies on two parts:

1. **Non-black-box reconstruction:** The reconstruction procedure answers queries of the learning algorithm to $f(x)$ by computing the mapping $(x,i) \mapsto f(x)_i$ in time T', relying on the fact that *it has explicit access to the input* x. With an appropriate setting of parameters, we can ensure that the reconstruction procedure runs in time noticeably less than $|f(x)| \cdot T'(n)$. (Specifically, we set $|f(x)| = n^{\Omega(\epsilon)}$ and have the PRG of [NW94] output only $|f(x)|^{\Omega(1)} = n^{\Omega(\epsilon)}$ pseudorandom bits; see details below.)

2. **Learning a small circuit $\Rightarrow$ batch-computing the truth-table:** Given the small oracle circuit C that the learning algorithm outputs, our reconstruction procedure can quickly print the entire string $f(x)$, by evaluating C^D on all inputs. Specifically, it can do so in time $|f(x)| \cdot \tilde{O}(|C|) \cdot T(n)$, which we can ensure is noticeably less than $|f(x)| \cdot T'(n)$, by defining T' to be slightly larger than T (see below). This contradicts the hardness of f on input x.

Assuming that $f(x)$ is hard to print in time $|f(x)| \cdot T'(n)$ on 0.99 of the inputs x, our derandomization succeeds on 0.99 of the inputs. In fact, for every polynomial-time samplable distribution $\mathbf{x}$, if $f(x)$ is hard to print with high probability over $x \sim \mathbf{x}$, then our derandomization succeeds with high probability over $x \sim \mathbf{x}$ (the requirement that $\mathbf{x}$ will be polynomial-time samplable is due to our use of one-way functions, which was not described above; we explain this in a moment). By tweaking the parameters, we can even relax the required lower bound to be of the form $|f(x)|^{\alpha} \cdot T'(n)$ for an arbitrarily small constant $\alpha > 0$, rather than $|f(x)| \cdot T'(n)$.

Unfortunately, the idea above does not work as-is. First, as mentioned above, to make this idea work we need the number of output bits of the PRG of [NW94; IW99] to be only $n^{\Omega(\epsilon)}$ rather than $T(n)$; we will use our assumption that one-way functions exist to bridge this gap (this assumption implies the existence of a very fast PRG with an arbitrarily large polynomial stretch). Secondly, the PRG of [NW94; IW99] actually encodes the truth-table $f(x)$ by error-correcting codes and the reconstruction procedure needs query access to the encoded string rather than to $f(x)$; indeed, these encodings do not preserve the time-complexity of computing individual bits of the string. We explain below how to overcome this difficulty, using the stronger assumption that $f(x)$ is hard to "approximately-print".

[12]These queries are used for the reconstruction procedure of the PRG of [NW94] as well as to eliminate the required advice in the list-decoding algorithms of [GL89; IW99], which are both used as part of the reconstruction procedure.

Implementation: Some technical details. Recall that our goal is to derandomize a probabilistic algorithm that runs in time $T(n)$. Fix any small constant $\alpha > 0$ (which will characterize the hardness of f), let $\epsilon' > 0$ be a sufficiently small constant, and let $k = n^{O(\epsilon'/\alpha)} \ll n^{\alpha}$. For $T' = T \cdot k$, we fix a function $f\colon \{0,1\}^n \to \{0,1\}^k$ such that $(x,i) \mapsto f(x)_i$ is computable in time T', but every probabilistic algorithm that runs in time $T' \cdot k^{\alpha}$ fails, on all but a small fraction of inputs x, to print whp a string $\widetilde{f}(x)$ that agrees with $f(x)$ on more than 0.99 of the bits. Recall that such a function f is obtained naturally as the k-wise direct-product of a function $f_0 \in \mathcal{DTIME}[T']$ that is hard for probabilistic time slightly smaller than T' (see Section 6.2 for details).

As a first step, relying on the assumption that one-way functions exist, we use a PRG that runs in near-linear time to reduce the number of random coins used by the probabilistic algorithm from $T(n)$ to $n^{\epsilon'}$, without significantly increasing its running time and while maintaining correctness on all but a negligible fraction of inputs (see Theorems 3.4 and 6.4 and Claim 6.4.1 for details). For simplicity, let us assume that we reduced the number of random coins to $n^{\epsilon'}$ without affecting the running time or correctness at all.

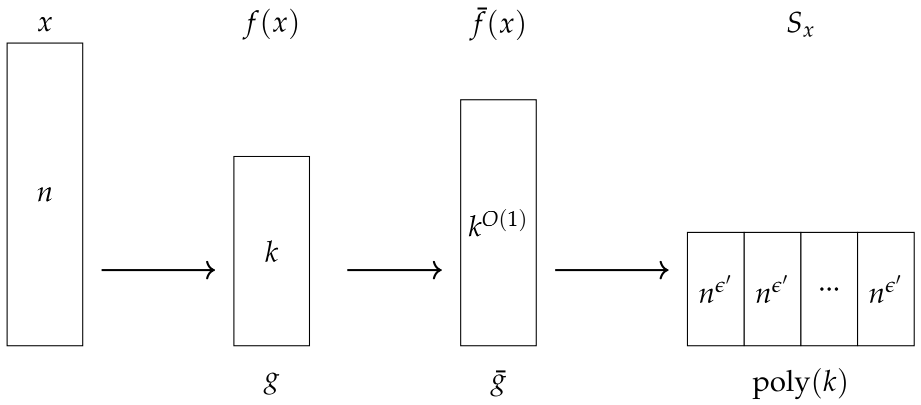
The targeted PRG. On input x we first compute the string $f(x)$, which we think of as the truth-table of a function $g\colon [k] \to \{0,1\}$. We then apply the "mild average-case" to "extreme average-case" hardness amplification step of [IW99] to the function g, and then apply the Hadamard encoding to the resulting non-Boolean function, to obtain the truth-table $\bar{f}(x)$ of a function $\bar{g}$. We instantiate the NW PRG with $\bar{g}$ as the hard function and with output length $n^{\epsilon'}$, and this yields a collection $S_x \subseteq \{0,1\}^{n^{\epsilon'}}$ of strings that we hope fools the probabilistic algorithm on input x.

The sequence of transformations $x \mapsto f(x) \mapsto \bar{f}(x) \mapsto S_x$ above is depicted visually in Figure 2.1. One crucial point for us is that the first two transformations, mapping x to $\bar{f}(x)$, satisfy the following property: Each output bit of $\bar{f}(x)$ can be computed, when given access to x, in time $T' \cdot \log(k)$. The reason is that each output bit of $f(x)$ is computable from x in time T' by our assumption, and the truth-table $\bar{f}(x)$ is obtained from $f(x)$ by applying the efficient derandomized direct product of [IW98] and the Hadamard encoding, which approximately maintain the complexity of the underlying function (see Theorem 6.3 and Appendix A.2 for details).

The resulting derandomization algorithm is extremely fast: We compute the string $f(x)$ in time $T' \cdot k$, transform it into $\bar{f}(x)$ and S_x in time $\mathrm{poly}(k)$, and evaluate the original probabilistic algorithm on each string in S_x (and output the majority value) in time $\mathrm{poly}(k) \cdot T' = \mathrm{poly}(k) \cdot T$. Choosing ϵ' to be sufficiently small, the running time is less than $n^{\epsilon} \cdot T$.

Analysis: Non-black-box reconstruction. Our goal is to design a reconstruction procedure that, when given access to x and to a time-T distinguisher D for S_x, runs in time $T' \cdot n^{\alpha}$ and with high probability prints a string $\widetilde{f}(x)$ that agrees with $f(x)$ on more than 0.99 of the bits. Our procedure will use the reconstruction algorithm `Rec` of the NW PRG as well as the list-decoding algorithm `Dec` underlying the transformation of g to $\bar{g}$ (the latter combines the list-decoding algorithms of the derandomized direct-product of [IW99] and of the Hadamard encoding [GL89]).

Since the output length of the targeted PRG is small (i.e., the output length is $n^{\epsilon'}$), both `Rec` and `Dec` run in time at most $n^{O(\epsilon')} \leq k^{\alpha/2}$ (see Theorem 6.3 for details). Whenever `Rec` queries $\bar{g} = \bar{f}(x)$ at location $i \in [\mathrm{poly}(k)]$, our reconstruction procedure uses its access to x in order to compute the mapping $(x,i) \mapsto \bar{f}(x)_i$ in time $T' \cdot \log(k)$. Also, the list-decoding algorithm `Dec` produces a list of candidate circuits, and we estimate the agreement of each of these circuits



$\Rightarrow$ Each entry $i \in [k]$ of $f(x)$ is computable from x in time T'.

$\Rightarrow$ Each entry $i \in [k^{O(1)}]$ of $\bar{f}(x)$ is computable from x in time $T' \cdot \log(k)$.

Figure 2: A diagram of the steps in our construction, noting the lengths of the strings in each step. Note that the function g is $\{0,1\}^{\log(k)} \to \{0,1\}$ and that the function $\bar{g}$ is $\{0,1\}^{O(\log(k))} \to \{0,1\}$. Also, the number of strings in S_x is $\mathrm{poly}(k)$, and each of them has $n^{\epsilon'}$ bits.

with $f(x)$ up to a small constant error, using random sampling and by computing the mapping $(x,i) \mapsto f(x)_i$, which can be done in time T' using our access to x.

After running this procedure, with high probability we obtain an oracle circuit C of size at most $k^{\alpha/2}$ such that the truth-table of C^D agrees with $f(x)$ on more than 0.99 of the inputs. We can then compute this truth-table in time $k \cdot \tilde{O}(|C|) \cdot T = \tilde{O}(|C|) \cdot T' < k^{\alpha} \cdot T'$ and print a string $\widetilde{f}(x)$ that agrees with $f(x)$ on more than 0.99 of the bits.

Two additional comments. The reconstruction procedure above is "instance-wise", in the sense that for every x such that the probabilistic algorithm distinguishes the outputs of the targeted PRG on x from uniform, the reconstruction procedure prints an approximate version of $f(x)$. Thus, when assuming hardness of f over the uniform distribution (as in Theorem 1.6) we obtain derandomization over the uniform distribution; and when hardness of f is over *all polynomial-time-samplable distributions* $\mathbf{x}$ (as in Theorem 1.7) we obtain a derandomization that succeeds over all polynomial-time-samplable distributions. Further details appear in Section 6.

As mentioned in Section 1.2, we also show that the existence of a "non-batch-computable" function is necessary for average-case superfast derandomization in the natural setting of highly-uniform balanced formulas. In high level, assuming that we can derandomize probabilistic formulas with overhead $T \mapsto T \cdot n^{\epsilon}$, we first diagonalize against all probabilistic formulas of size T using a function that is computable by a formula F of size $T' = T \cdot n^{\epsilon}$. We define a non-batch-computable function $g\colon \{0,1\}^n \to \{0,1\}^{n^{2\epsilon}}$ that, on input x, prints the $n^{2\epsilon}$ gate values in the appropriate intermediate level of $F(x)$. Due to our choice of computational model (i.e., balanced formulas) each output bit of g can be computed in time approximately $T'/n^{2\epsilon} = T \cdot n^{-\epsilon}$; whereas probabilistically computing the gate-values of the entire layer in F in time $o(T)$ allows to compute F itself in time $o(T)$, a contradiction to its hardness. See Proposition 6.15 for further details.

2.2 The proof of Theorems 1.2, 1.3, 1.4 and 1.5

Loosely speaking, in Section 2.1 we used the input x to produce a hard truth-table $f(x)$, and then instantiated a version of the NW PRG with $f(x)$ as the hard function. The main technical bottleneck was that the reconstruction procedure for the PRG needed oracle access to $f(x)$, and we were able to answer the procedure's oracle queries by our assumption that the individual bits of $f(x)$ are computable quickly. In Theorem 1.2 we have no such assumption, and therefore we will need a more complicated construction.

The classical reconstruction approach of [IW98] and its drawbacks. Previous works following [IW98; TV07] handled a similar technical challenge using an influential "bootstrapping" argument. Recall that this argument instantiates a version of the NW PRG using a hard problem $L \in \mathcal{PSPACE}$ that is both downward self-reducible and randomly self-reducible. Assuming that there exists a distinguisher for the PRG on *all input lengths* $n \in \mathbb{N}$, the reconstruction procedure is *iterative*: For $i = 1, ..., n$, the i^{th} reconstruction step constructs a circuit that decides $L_i = L \cap \{0,1\}^i$, using the circuit for L_{i-1} and the distinguisher for NW^{L_i} (i.e., for the NW PRG that uses L_i as the hard function). The base case of this procedure (i.e., computing L_1) is trivial, each iterative step uses the self-reducibility properties of L, and the conclusion of this argument is that L_n can be efficiently decided for every $n \in \mathbb{N}$, a contradiction to the hardness of L.

There are two well-known drawbacks in this argument. First, the entire argument is "black-box", in the sense that it does not involve the input x that is available to the PRG and to the reconstruction procedure. This is a drawback (rather than an advantage) since it causes their derandomization to *succeed only in average-case*, rather than in the worst-case.[13] Secondly, the argument yields derandomization that *succeeds only on infinitely many input lengths*, since the assumption that we make towards a contradiction is that a distinguisher succeeds (in distinguishing the PRG from uniform) on almost all input lengths.

A non-black-box version of their reconstruction argument. The main building-block in our proof is a non-black-box version of the [IW98] reconstruction argument, which allows us to overcome the two aforementioned drawbacks. Intuitively, for any fixed $x \in \{0,1\}^n$, instead of using the truth-tables of $L_1, ..., L_n$ as hard functions,[14] we use a *sequence of truth-tables that can be efficiently produced from the input* x, while guaranteeing that this sequence of truth-tables has the structural properties needed for the reconstruction argument. The only downside to our argument is that we construct a targeted HSG, rather than a targeted PRG. Details follow.

We first define a property of the hard function $f\colon \{0,1\}^n \to \{0,1\}^n$ that will allow us to construct a targeted reconstructive HSG using f. For parameters $t \ll T$ and $d \ll T$, we say that

[13]Specifically, their reconstruction procedure is a uniform algorithm that samples an input $x \in \{0,1\}^n$, and uses a corresponding distinguisher D_x for the PRG (that is obtained by running the probabilistic algorithm that we are trying to derandomize on input x) to compute L; since L is hard for probabilistic algorithms, it follows that the probability of sampling x on which the derandomization fails is small. When trying to strengthen the conclusion and deduce that there *does not exist* x on which the derandomization fails, we need to allow the reconstruction procedure access to an *arbitrary string* x. In such a case the reconstruction is a non-uniform procedure (the advice modeled by x), and so the argument does not work under the original uniform hardness hypothesis.

[14]To be accurate, in the arguments of [IW98; TV07] and of subsequent works the PRG uses the truth-tables of $L_1, ..., L_\ell$, where ℓ is proportional to the seed length of the PRG.

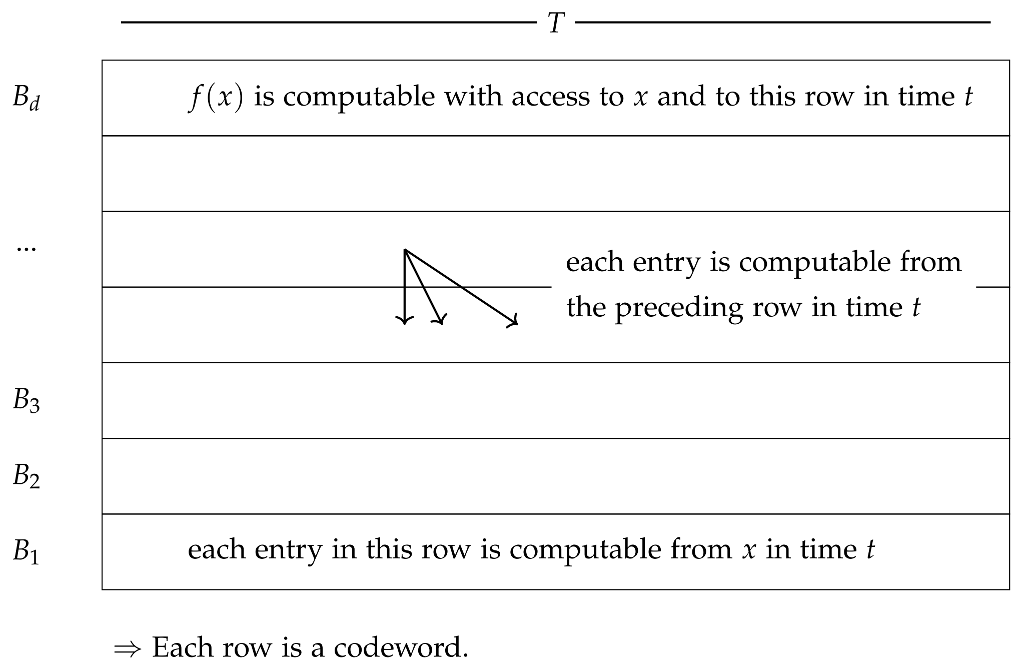

T
B_d
$f(x)$ is computable with access to x and to this row in time t
...
each entry is computable from the preceding row in time t
B_3
B_2
B_1
each entry in this row is computable from x in time t
$\Rightarrow$ Each row is a codeword.

Figure 3: Visual depiction of a $(d \times T)$-bootstrapping system $B = B(f, x)$. We denote the i^{th} row of B by B_i.

f has $(d \times T)$-bootstrapping systems with bootstrapping time t if for every $x \in \{0,1\}^n$ there exists a $d \times T$ matrix $B = B_f(x)$ satisfying the following.

1. **(Base case.)** There is an algorithm that gets input $(x, i) \in \{0,1\}^n \times [T]$, runs in time t, and outputs the i^{th} bit in the bottom row of B.

2. **(Final case.)** There is an algorithm that gets input x and oracle access to the top row of B, and outputs $f(x)$ in time $|f(x)| \cdot t$.

3. **(Self-reducibility.)** There is an algorithm that gets input $(i, j) \in [d] \times [T]$ and oracle access to the $(i-1)^{th}$ row of B, runs in time t, and outputs $B_{i,j}$.

4. **(Error-correction.)** Each row of B is a codeword in a sufficiently good code. (The entries in the matrix will be non-Boolean, but for simplicity we ignore this issue in the current overview.) In particular, a sufficient requirement from the code is that every row, considered as a truth-table, is sample-aided worst-case to rare-case reducible in time t.[15]

See Figure 3 for a visual depiction of a bootstrapping system, and see Definition 4.1 for the formal definition. We stress that the bootstrapping matrix $B = B_f(x)$ *depends on the particular input* x. In fact, if f has $(d \times T)$ bootstrapping systems with bootstrapping time t, then there is an efficient algorithm that maps x to $B_f(x)$ in time $\text{poly}(T, d, t)$ (i.e., the algorithm iteratively computes each row in $B_f(x)$ from bottom to top, using the self-reducibility algorithm).

[15]This property of a function g means that there is a probabilistic algorithm that gets input z, access to a highly corrupted version of g, and uniform samples $(r, g(r))$, and outputs $g(z)$ with high probability (see Definition 3.8, following Goldreich and Rothblum [GR17]). We can also relax this property, and only require that each row will be a codeword in a locally-decodable code; see the discussion after the proof of Proposition 4.2.

Previous works following [IW98; TV07] can also be viewed as using bootstrapping systems, albeit of a particular and more constrained type than the notion that we leverage (intuitively, the bootstrapping systems implicit in their works are "black-box" since they do not depend on the input; see Section 4.1 for an explanation). Using our more general notion, given any function that has $(d \times T)$-bootstrapping systems with bootstrapping time t satisfying $\mathrm{poly}(d,t) \ll T$, we can construct a corresponding reconstructive targeted HSG H_f that has the following parameters:

Proposition 2.1 (from bootstrapping systems to a targeted HSG; informal, see Proposition 4.4)**.** *Assume that $f\colon \{0,1\}^n \to \{0,1\}^n$ has $(d \times T)$-bootstrapping systems with bootstrapping time t and sufficiently good error-correction properties. Then, for any $\epsilon > 0$ and $n \le M(n) \le \min\left\{T(n)^{\Omega(\epsilon)}, t(n)^{\Omega(1)}\right\}$ there exist:*

1. **Targeted HSG.** *A deterministic algorithm H_f that gets input $x \in \{0,1\}^n$, runs in time $\mathrm{poly}(T)$, and outputs a set of M-bit strings.*
2. **Reconstruction procedure.** *A probabilistic algorithm R that gets input x and oracle access to a $(1/M)$-distinguisher D (i.e., $\Pr_{r\in\{0,1\}^M}[D(r) = 1] \ge 1/M$ but D rejects all the strings that $H_f(x)$ outputs), runs in time $\mathrm{poly}(t, T^\epsilon, M) \cdot d$, and with probability at least $2/3$ outputs $f(x)$.*

We explain the proof of Proposition 2.1, which uses the same ideas as in [IW98] but now applies them to the bootstrapping system $B_f(x)$ that depends on x (rather than to a sequence of the truth-tables of a hard function on input lengths $1, ..., n$). The targeted HSG thinks of each row in $B_f(x)$ as the truth-table of a function, and applies a suitable instantiation of the NW PRG to this truth-table in order to map it to a set of M-bit strings.[16] The union over all $i \in [d]$ of these sets of M-bit strings is the final pseudorandom set of strings that our targeted HSG outputs.

The reconstruction procedure R gets input x and access to a distinguisher D_x for $H_f(x)$, and iteratively constructs small circuits that compute each row of $B_f(x)$, from bottom to top. The first circuit computes the "base case" – the bottom row in the matrix $B_f(x)$. This can be done by a circuit of size $\tilde{O}(t)$ since the reconstruction procedure has access to x. Then, for $i \ge 2$, we mimic the iterative reconstruction argument of [IW98], showing that when given access to a circuit C_{i-1} that computes the truth-table B_{i-1}, and to the distinguisher D_x, we can compute in time $\mathrm{poly}(t, M)$ a circuit C_i that computes the truth-table B_i. (In a gist, this step consists of combining the well-known reconstruction and list-decoding algorithms of [GL89; NW94; IW98] and the sample-aided worst-case to rare-case reducibility algorithm for B_i, which in our construction follows [STV01]; see Section 4.4 for details.) Finally, we obtain a circuit C_d that computes B_d, which allows us to compute $f(x)$ by evaluating the circuit $|f(x)| \cdot t$ times. The crucial point is that the reconstruction time is a fixed polynomial in M, d and t, which can be much smaller than T assuming that $t \ll T$ and $d \ll T$.

Indeed, the description above is high-level and omits many technical details, but these generally follow known techniques. The formal connection between bootstrapping systems and HSGs is stated in Proposition 4.4 and proved in Section 4.4.

[16]For simplicity, in this presentation we simply refer "a suitable version" of the NW PRG, ignoring the alphabet from which each entry in the matrix comes from and additional encodings applied to each row.

Bootstrapping systems and logspace-uniform circuits of bounded depth. So far we proved a generic statement, saying that if f has bootstrapping systems then we can design a targeted reconstructive HSG using f as the hard function. Loosely speaking, the additional result needed to prove Theorem 1.2 is that *the class of functions that have bootstrapping systems with $d, t \ll T$ is essentially the class of functions computable by logspace-uniform circuits of bounded depth.*

One direction in this connection is relatively straightforward: If f has $(d \times T)$-bootstrapping systems with bootstrapping time t, then f can be computed by circuits of depth approximately $(d \cdot t)$ and size approximately $(d \cdot t) \cdot T$ (this direction is visually evident from Figure 3; see Proposition 4.2 for details). The other direction is the more demanding one: We show that any function computable by logspace-uniform circuits of depth d and size T also has bootstrapping systems with dimensions $d \cdot \log(T) \times \text{poly}(T)$ and bootstrapping time T^{ϵ}.

Proposition 2.2 (bootstrapping systems for logspace-uniform circuits; informal, see Proposition 4.3)**.** *Let $f \colon \{0,1\}^n \to \{0,1\}^n$ be computable by logspace-uniform circuits of size $T(n)$ and depth $d(n)$ and let $\epsilon > 0$ be an arbitrarily small constant. Then, for $d' = d \cdot \log(T)$ and $T' = \text{poly}(T)$ there are $(d' \times T')$-bootstrapping systems for f with bootstrapping time $t = T^{\epsilon} \cdot n$ and sufficiently good error-correction properties.*

To get initial intuition for the proof, assume for a moment that f from Proposition 2.2 is computable by circuits that are not only logspace-uniform, but such that given the index of a gate g in the circuit, we can compute the indices of the two gates feeding into g in time $n^{o(1)}$. Then, it is easy to construct a $(d \times T)$ system $B = B_f(x)$ that satisfies the base case, the final case, and the self-reducibility properties of a bootstrapping system, but does not necessarily satisfy any error-correction property. Specifically, denoting the circuit for f by C_f, and considering the gate-values of C_f when it is given input x, we can define each row in B to be the gate-values in the corresponding row of $C_f(x)$. The self-reducibility property then follows since the value of each gate in the i^{th} row of $C_f(x)$ depends on the values of just two gates in the $(i-1)^{th}$ row of $C_f(x)$, and we can compute the indices of these gates in time $n^{o(1)}$. The challenge in proving Proposition 2.2 is to construct a system in which the rows are not only self-reducible, but are simultaneously also error-correctable (i.e., codewords in a good code), and to do so even when the circuit only satisfies the weaker logspace-uniformity property.

To do so we rely on the ideas underlying the doubly-efficient proof system of Goldwasser, Kalai, and Rothblum [GKR15]. Let us recall their construction. For $i = 1, ..., d$, denote by $\alpha_i \in \{0,1\}^T$ the string representing the gate-values in the i^{th} row of $C_f(x)$ (note that we index the bottom row by 1 and the top row by d). Thinking of each α_i as a function $\{0,1\}^{\log(T)} \to \{0,1\}$, let $\hat{\alpha}_i$ be a suitable low-degree extension of α_i over a field of sufficiently large polynomial size (the precise degree of each $\hat{\alpha}_i$ will be T^{ϵ} for a small constant $\epsilon > 0$).

As a first step (which does not yet complete the construction), define a system $\widetilde{B}$ in which the i^{th} row is the truth-table of $\hat{\alpha}_i$ (i.e., the evaluation of the polynomial $\hat{\alpha}_i$ on all inputs). Indeed, low-degree polynomials are codewords in the Reed-Muller code, and this code satisfies the error-correction properties that we need in a bootstrapping system (see Section 3.4). However, we now need to show that the rows in the system are self-reducible.

The key technical idea in [GKR15] is an interactive protocol that reduces computing $\hat{\alpha}_{i+1}$ at any given point $\vec{w}$ to computing $\hat{\alpha}_i$ at a different point (that the verifier chooses). This interactive reduction consists of running an appropriate sumcheck protocol on a low-degree polynomial that expresses the value $\hat{\alpha}_{i+1}(\vec{w})$ as the weighted sum of values of $\hat{\alpha}_i$ on T^2 inputs. Specifically, let $\hat{\Phi}$ be an appropriate arithmetization of the circuit-structure function (i.e., of the function that gets input (w, u, v) and outputs 1 iff gate w is fed by gates u and v), and assume wlog that all gates in C_f compute the NAND function. Then, we have that

$$\hat{\alpha}_{i+1}(\vec{w}) = \sum_{\vec{u},\vec{v}} \hat{\Phi}(\vec{w},\vec{u},\vec{v}) \cdot (1 - \hat{\alpha}_i(\vec{u}) \cdot \hat{\alpha}_i(\vec{v})) \ , \tag{2.1}$$

where the summation ranges over the arithmetizations $\vec{u}, \vec{v}$ of all T^2 pairs of gates u, v in the i^{th} layer. (It is not a-priori clear how to arithmetize the circuit-structure function by a polynomial $\hat{\Phi}$ of a low degree. This problem was solved in [GKR15], relying on the assumption that C_f is logspace-uniform, using another auxiliary interactive protocol; we avoid this auxiliary protocol relying on a simplification suggested by Goldreich [Gol18]. See Section 4.3 for details.)

The sumcheck protocol reduces computing the LHS in Eq. (2.1) to computing $\hat{\alpha}_i$ at two points, in at most $\log(T)$ rounds.[17] The last observation that we need to complete the proof of Proposition 2.2 dates back to the work of Trevisan and Vadhan [TV07]: They observed that the sumcheck protocol can be thought of as yielding a sequence of polynomials that is self-reducible, in the sense that computing each polynomial reduces to computing the subsequent polynomial in a small number of points. (Each polynomial corresponds to a round in the protocol, and the number of points corresponds to the degree of the original polynomial in Eq. (2.1).)

We are now ready to define the bootstrapping system. We start from the initial system $\widetilde{B}$ whose layers are the $\hat{\alpha}_i$'s, and in between each pair of layers we add the polynomials that arise from the sumcheck protocol. The self-reducibility property now follows since the polynomials in each pair of layers either both come from the sumcheck protocol (in which case they are self-reducible by the observation of [TV07]), or one of them is the last one in the sequence of polynomials corresponding to $\hat{\alpha}_{i+1}$ and the other one represents $\hat{\alpha}_i$ (in which case the former polynomial just depends on the values of the latter in two points). For the full proof details see Proposition 4.3 and Section 4.3.

An alternative view of our construction. An equivalent way of viewing our construction is that each row in the bootstrapping system is the truth-table of the prover's strategy function on input x in a corresponding round of the [GKR15] protocol. Specifically, the d^{th} row (i.e., the top row) corresponds to the first round, in which the prover just sends the claimed value $f(x)$; the $(d-i)^{th}$ row corresponds to round $i+1$ in the proof system; and the bottom row corresponds to the last round in the proof system. From this perspective, the reconstruction procedure for B that was described above reconstructs the prover strategy functions in the protocol of [GKR15], round-by-round, starting from the last round in the interaction and working back until the first round.

Wrapping-up: Proof of Theorem 1.2 Combining Propositions 2.1 and 2.2, if f has logspace-uniform circuits of size $T(n) = \text{poly}(n)$ and depth $d(n) = n^2$, then we have a reconstructive targeted HSG H_f with output length $M = O(n) = T^{\Omega(1)}$ and reconstruction time $\bar{t} = \text{poly}(T^{\epsilon}, n)$.

[17] In fact, since we use low-degree extensions $\hat{\alpha}_i$ of degree $T^{\Omega(1)}$, the number of rounds will be constant. However, this is immaterial to the high-level overview and we mention it only to avoid confusion.

In particular, if there exists an f as above that cannot be computed by probabilistic algorithms in time $\bar{t}$ on *almost all inputs* x, then for every x we have that H_f "fools" all linear-time machines that also get access to x, and it follows that $pr\mathcal{RP} = pr\mathcal{P}$. The standard reductions of [Sip83; Lau83] imply that $pr\mathcal{BPP} = pr\mathcal{P}$, concluding the proof of Theorem 1.2.

We wish to highlight where we used the assumption that f is computable in parallel, i.e. by uniform circuits of low depth. Denoting the circuit depth by $d = d(n)$, we construct (via Proposition 2.2) a bootstrapping system of slightly larger depth $d' > d$, although for now we can pretend that $d' = d$. The reconstruction algorithm from Proposition 2.1 then runs in d' iterations, and in each iteration it performs a computation in time $\text{poly}(n)$ (when setting the parameters M, T^{ϵ} and t as above). Thus, our hardness assumption is for probabilistic algorithms that run in time $d' \cdot \text{poly}(n)$. In particular, this means that the deterministic time bound for computing f must be noticeably larger than the depth d' of the bootstrapping system and of the circuit for f.

We also comment that to deduce the derandomization conclusion, it is not actually necessary to assume that f is hard for *all* probabilistic algorithms that run in the prescribed time; instead, it suffices to assume that f is hard for a *particular* algorithm. See Remark 5.6 for details.

Extensions: Proofs of Theorems 1.3, 1.4 and 1.5. The generalization of Theorem 1.2 to Theorem 1.3 (i.e., scaling the time bounds for the hard function and the derandomization algorithm) is mainly based on parametric modifications to the argument, and does not require additional new ideas. However, one interesting point is that we reduce derandomization of $pr\mathcal{BPTIME}$ to derandomization of $pr\mathcal{RTIME}$ with a significant time overhead (using the classical techniques of [Sip83; Lau83], following [BF99]); more efficient reductions are known (see [ACR98; GVW11]), but these work only when the derandomization of $pr\mathcal{RTIME}$ is black-box (using HSGs), whereas our derandomization is non-black-box.

Scaling Theorem 1.2 to smaller circuit classes, and in particular to logspace-uniform $\mathcal{NC}$ as in Theorem 1.5, requires significantly more work and involves many low-level technical details. Given a function f from the class, we need to ensure that the construction of bootstrapping system, targeted HSG, and reconstruction procedure can all be carried out by circuits that are both of *polylogarithmic depth* and *logspace-uniform*. To do so we prove several technical results that are seemingly new (although they rely on known high-level algorithmic ideas); one result that might be of independent interest is that the list-decoding procedure for the Reed-Muller code can be implemented by such circuits, since the corresponding special case of list-decoding the Reed-Solomon code can be implemented by such circuits. This construction relies on an idea that was communicated to us by Madhu Sudan, and is presented in Appendix B.

Lastly, let us explain how we prove the "low-end" hardness-to-randomness tradeoff in Theorem 1.4, in which there are *no structural restrictions* on the hard function f. First, if $\mathcal{EXP} \not\subset \mathcal{P}/\text{poly}$ then $pr\mathcal{BPP} \subseteq \texttt{i.o.}pr\mathcal{SUBEXP}$ (using the NW PRG, see [BFN+93]) and we are done. Otherwise, by [BFN+93], we have that $\mathcal{EXP} = \mathcal{MA}$. In particular, the function f that is computable in time 2^{n^c} has $\mathcal{MA}$ protocols running in some *fixed* time bound n^k. The crucial observation is that any such $\mathcal{MA}$ protocol can by simulated by a uniform circuit that, while having size $2^{O(n^k)}$, is nevertheless of *fixed polynomial depth* $d(n) = n^{O(k)}$; this is by a brute-force circuit that enumerates the witnesses for the $\mathcal{MA}$ verifier *in parallel*. Moreover, this circuit is logspace-uniform, since we just use the standard transformation of Turing machines to highly-uniform circuits that compute the tableau of the machine's computation.

We now want to base a reconstructive targeted HSG H_f on this hard function f. To do so we need to tweak the parameters in Propositions 2.1 and 2.2 so that the reconstructive overhead and bootstrapping time will both be proportional to $\text{polylog}(T) = \text{poly}(n)$ rather than to $T^{\epsilon} = 2^{\text{poly}(n)}$. This modification is indeed possible, where the cost is that the dimensions of the bootstrapping system increase and the resulting targeted HSG is slower, working in time $2^{\text{polylog}(T)} = 2^{n^{O(c)}}$ (see Propositions 4.3 and 4.4 for details). This overhead is fortunately good enough for the current setting: Instantiating the targeted HSG H_f with such overhead and with polynomial output length M, the derandomization time is a fixed exponent $2^{n^{O(c)}}$ and the reconstruction time is $\text{polylog}(T) \cdot \text{poly}(n) \cdot d = \text{poly}(n)$. Thus, if f is hard for probabilistic polynomial-time algorithms then $pr\mathcal{RP}$ can be simulated in fixed exponential time $2^{n^{O(c)}}$.

3 Preliminaries

Throughout the paper we use the notation $\langle x, y\rangle$ to denote the inner-product over $\mathbb{F}_2$; that is, $\langle x, y\rangle = \oplus_i (x_i \cdot y_i)$. We will typically denote random variables by boldface.

Unless explicitly stated otherwise, we assume that all circuits are comprised of Boolean `NAND` gates of fan-in two and are layered, where the latter property means that gates at distance i from the inputs only feed to gates at distance $i+1$. [18] In several places in the paper we will need the following notion, which strengthens the standard notion of a time-computable function by requiring that the function will be computable in logarithmic space.

Definition 3.1 (logspace-computable functions)**.** *We say that a function $T\colon \mathbb{N} \to \mathbb{N}$ is* logspace-computable *if there exists an algorithm that gets input 1^n, runs in space $O(\log(T(n)))$, and outputs $T(n)$.*

3.1 Pseudorandomness and targeted pseudorandom generators

Our non-black-box derandomization algorithms rely on constructions of *targeted pseudorandom generators* and *targeted hitting-set generators*. Targeted PRGs were defined by Goldreich [Gol11a], who showed that the existence of polynomial-time computable targeted PRGs with logarithmic seed length is equivalent to the hypothesis $pr\mathcal{BPP} = pr\mathcal{P}$. Let us recall his definition of targeted PRGs, and also define a targeted HSGs in the natural way:

Definition 3.2 (targeted PRG; see [Gol11a, Definition 4.10])**.** *For $T\colon \mathbb{N} \to \mathbb{N}$, let G be an algorithm that gets input $x \in \{0,1\}^n$ and a seed of length $\ell(n)$ and outputs a string of length $T(n)$. We say that G is a* targeted pseudorandom generator with error μ for time T *(or μ-targeted-PRG for time T, in short) if for every algorithm A running in time T, every sufficiently large $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$ it holds that*

$$\left| \Pr_{r\in\{0,1\}^{T(n)}}[A(x,r) = 1] - \Pr_{s\in\{0,1\}^{\ell(n)}}[A(x,G(x,s)) = 1] \right| \leq \epsilon\ .$$

[18] The assumption that circuits are layered is made merely for simplicity. In particular, logspace-uniform circuits (as defined in Definition 3.5) that are not layered can be transformed to logspace-uniform circuits that are layered with a linear overhead in size. (By adding dummy gates at each layer, and since the distance of each gate from the inputs can be computed in space that is logarithmic in the size of the circuit.)

Definition 3.3 (targeted HSG)**.** *For $T\colon \mathbb{N} \to \mathbb{N}$, let H be an algorithm that gets input $x \in \{0,1\}^n$ and a random seed of length $\ell(n)$ and outputs a string of length $T(n)$. We say that H is a* targeted hitting-set generator with error μ for time T *(or μ-targeted-HSG for time T, in short) if for every algorithm A running in time T, every sufficiently large $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$ such that $\Pr_{r\in\{0,1\}^{T(n)}}[A(x,r) = 1] \geq 1/2$, there exists $s \in \ell(n)$ such that $A(x, H(x,s)) = 1$.*

The foregoing definitions refer to PRGs and HSGs that fool distinguishers in the worst-case (i.e., for every $x \in \{0,1\}^n$). These definitions extend naturally to PRGs and HSGs that only fool distinguishers on average-case (i.e., with high probability over choice of x according to some predetermined probability distributions).

In Section 6 we rely on the following claim, which asserts that if one-way functions exist, then there exist PRGs with seed length n^ϵ that are computable in time $n^{1+\epsilon}$, for an arbitrarily small $\epsilon > 0$. The proof of this claims amounts to using the classical constructions of PRGs from one-way functions [HIL+99] and then applying standard techniques to extend the expansion factor of PRGs (see, e.g., [Gol01, Construction 3.3.2]).

Theorem 3.4 (OWFs yield PRGs with near-linear running time)**.** *If there exists a polynomial-time computable one-way function secure against polynomial-time algorithms, then for every $\epsilon > 0$ there exists a PRG that has seed length $\ell(n) = n^\epsilon$, is computable in time $n^{1+\epsilon}$, and fools every polynomial-time algorithm with negligible error.*

Proof. By [HIL+99], the hypothesis implies that for some negligible function neg there exists $G_1\colon \{0,1\}^m \to \{0,1\}^{2m}$ that is computable in time m^c, and for all $k \in \mathbb{N}$, no probabilistic algorithm running in time $(2m)^k$ can distinguish between $G_1(\mathbf{u}_m)$ and $\mathbf{u}_{2m}$ with advantage at least $\mathsf{neg}(2m)$. Our PRG G gets input 1^n and a seed $x \in \{0,1\}^m$ where $m = n^{\epsilon/2c} < n^\epsilon$ and acts as follows.

1. Let $\sigma_1 = x$.
2. For $i \in \{2, 3, ..., n/m\}$, compute σ_i as the last m bits of $G_1(\sigma_{i-1})$.
3. Output the concatenation of $\sigma_1, \sigma_2, ..., \sigma_{n/m}$, which is an n-bit string

The running time of G is at most $m^c \cdot n + O(n) \leq n^{1+\epsilon}$, and a standard hybrid argument (as in [Gol01, Theorem 3.3.3]) reduces distinguishing G_1 with noticeable advantage to distinguishing G with noticeable advantage. ■

3.2 Logspace-uniform circuits

Recall that, as mentioned in the introduction, we generalize the definition of logspace-uniform circuits to super-polynomial size bounds, in a straightforward way. The usual definition refers to circuits of size $\mathrm{poly}(n)$ whose adjacency relation (i.e., $\Phi(u, v, w) = 1$ iff gates v, w feed into gate u) can be decided in space $O(\log(n))$; that is, in space that is linear in the input size to the adjacency relation function. We simply scale this definition up, while maintaining the requirement that the adjacency relation can be decided in space logarithmic in the circuit size (i.e., linear in the input size to the adjacency relation); that is:

Definition 3.5 (logspace-uniform circuit). *We say that a circuit family $\{C_n\}_{n\in\mathbb{N}}$ of size $T(n)$ is* logspace-uniform *if there exists an algorithm A such that:*

1. *(**Decides the adjacency relation.**) The algorithm gets as input $(u,v,w)\in\{0,1\}^{3\log(T(n))}$ and accepts if and only if the gates in C_n indexed by v and by w feed into the gate in C_n indexed by u.*
2. *(**Runs in linear space.**) On an input of length $\ell = 3\log(T(n))$, the algorithm runs in space $O(\ell)$.*

When we mention logspace-uniform probabilistic circuits *we refer to circuits that are logspace-uniform and also use additional input gates that are assigned random values (i.e., the randomness is used by the circuit rather than by the logspace algorithm that constructs the circuit).*

Note that Definition 3.5 is equivalent to a definition asserting that there exists an algorithm that gets input 1^n, runs in space $O(\log(T))$, and prints a description of C_n (where the description is a list of gates, and for each gate we list the indices of gates feeding into it).

3.3 Average-case complexity classes and simulations

We now recall standard definitions of average-case simulation of a problem $L\subseteq\{0,1\}^*$. The following definitions refer to average-case simulation over an arbitrary distribution, over the uniform distribution, and over all polynomial-time samplable distributions, respectively.

Definition 3.6 (average-case simulation over arbitrary distributions). *Let $L\subseteq\{0,1\}^*$, let $\beta\colon\mathbb{N}\to(0,1)$ let $\mathcal{C}$ be a complexity class, and let $\mathbf{x}=\{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be an ensemble of distributions (where $\mathbf{x}_n$ is a distribution over n-bit inputs). We say that $L\in\mathsf{heur}_{\mathbf{x},1-\beta}\text{-}\mathcal{C}$ if there exists $C\in\mathcal{C}$ such that for every sufficiently large $n\in\mathbb{N}$ it holds that $\Pr_{x\sim\mathbf{x}_n}[C(x)=L(x)]\geq 1-\beta(n)$.*

Definition 3.7 (average-case simulation, two special cases). *Let $L\subseteq\{0,1\}^*$, let $\beta\colon\mathbb{N}\to(0,1)$ and let $\mathcal{C}$ be a complexity class. Then:*

1. *We say that $L\in\mathsf{avg}_{1-\beta}\text{-}\mathcal{C}$ if there exists $C\in\mathcal{C}$ such that for every sufficiently large $n\in\mathbb{N}$ it holds that $\Pr_{x\in\{0,1\}^n}[C(x)=L(x)]\geq 1-\beta(n)$ (i.e., the choice of x is according to the uniform distribution).*
2. *We say that $L\in\mathsf{heur}_{1-\beta}\text{-}\mathcal{C}$ if there exists $C\in\mathcal{C}$ such that for every polynomial-time samplable ensemble $\mathbf{x}$ of distributions and sufficiently large $n\in\mathbb{N}$ it holds that $\Pr_{x\sim\mathbf{x}_n}[C(x)=L(x)]\geq 1-\beta(n)$.*

We say that a complexity class $\mathcal{C}'$ satisfies $\mathcal{C}'\subseteq\mathsf{avg}_{1-\beta}\text{-}\mathcal{C}$ if for every $L\in\mathcal{C}'$ it holds that $L\in\mathsf{avg}_{1-\beta}\text{-}\mathcal{C}$ (and similarly define $\mathcal{C}'\subseteq\mathsf{heur}_{\mathbf{x},1-\beta}\text{-}\mathcal{C}$ and $\mathcal{C}'\subseteq\mathsf{heur}_{1-\beta}\text{-}\mathcal{C}$).

3.4 Sample-aided worst-case to rare-case reductions for polynomials

Our algorithms will use sample-aided worst-case to rare-case reductions. The foregoing term was coined recently by Goldreich and Rothblum [GR17], and is implicit in many previous works. Intuitively, a sample-aided worst-case to rare-case reduction for a function f is an algorithm that computes f at any point given a "highly corrupted" version $\widetilde{f}$ of f as well as random labeled examples of f. We also specialize this definition to the case where the algorithm can be implemented by logspace-uniform circuits of bounded depth.

Definition 3.8 (sample-aided reductions)**.** *For $s, w \colon \mathbb{N} \to \mathbb{N}$ and $\rho, \epsilon \colon \mathbb{N} \to [0,1]$:*

1. *Let $f \colon \{0,1\}^* \to \{0,1\}^*$ be a function that maps n bits to $w(n)$ bits.*
2. *Let M be a probabilistic procedure that gets input 1^n and a sequence of $s(n)$ pairs of the form $(r,v) \in \{0,1\}^n \times \{0,1\}^n$ and oracle access to a function $\widetilde{f}_n \colon \{0,1\}^n \to \{0,1\}^{w(n)}$, and outputs a circuit $C \colon \{0,1\}^n \to \{0,1\}^{w(n)}$ with oracle gates.*

We say that M is a sample-aided reduction of computing f in the worst-case to computing f on ρ of the inputs using a sample of size s and with error ϵ *if for every $\widetilde{f}_n \colon \{0,1\}^n \to \{0,1\}^{w(n)}$ satisfying $\Pr_{x\in\{0,1\}^n}\left[\widetilde{f}_n(x) = f_n(x)\right] \geq \rho(n)$ the following holds: With probability at least $1-\epsilon$ over choice of $\bar{r} = r_1, ..., r_{s(n)} \in \{0,1\}^n$ and over the internal coin tosses of M, we have that $M^{\widetilde{f}_n}(1^n, (r_i, f_n(r_i))_{i\in[s(n)]})$ outputs an oracle circuit C such that $Pr\left[C^{\widetilde{f}_n}(x) = f_n(x)\right] \geq 2/3$ for every $x \in \{0,1\}^n$.*

Definition 3.9 (sample-aided worst-case to rare-case reducibility)**.** *For $\rho, \epsilon \colon \mathbb{N} \to (0,1)$, and $T, D, s \colon \mathbb{N} \to \mathbb{N}$, we say that a function $f \colon \{0,1\}^* \to \{0,1\}^*$ is* sample-aided worst-case to ρ-rare-case reducible by logspace-uniform circuis of size T and depth D with error ϵ and sample size s *if there exists a sample-aided reduction M of computing f in worst-case to computing f on ρ of the inputs such that M uses $s(n)$ samples and has error $\epsilon > 0$, and can be implemented by a logspace-uniform circuits of size $T(n)$ and depth $D(n)$.*

The following result asserts that low-degree polynomials (i.e., the Reed-Muller code) are sample-aided worst-case to rare-case reducible. The proof of the foregoing statement is quite standard, but in the following result we will also assert something stronger: We claim that the reduction can be implemented by logspace-uniform circuits of polylogarithmic depth (i.e., in logspace-uniform $\mathcal{NC}$). A reduction meeting this additional efficiency requirement, and in particular the logspace-uniformity of the circuits, seems not to have been known before. The full proof involves many low-level details, and we present it in Appendix B. A key idea in the proof was suggested to us by Madhu Sudan.

Proposition 3.10 (low-degree polynomials are uniformly sample-aided worst-case to average-case reducible)**.** *Let $q \colon \mathbb{N} \to \mathbb{N}$ be a function mapping integers to primes, let $\ell \colon \mathbb{N} \to \mathbb{N}$ such that $n \geq \ell(n) \cdot \log(q(n))$, and let $d \colon \mathbb{N} \to \mathbb{N}$. Let $f = \{f_n\}_{n\in\mathbb{N}}$ be a sequence of functions such that f_n computes a polynomial $\mathbb{F}_n^{\ell(n)} \to \mathbb{F}_n$ of degree $d(n)$ where $|\mathbb{F}_n| = q(n)$. Then f is sample-aided worst-case to ρ-rare-case reducible by logspace-uniform oracle circuits of size* $\mathrm{poly}(q,\ell)$ *and depth* $\mathrm{polylog}(q,\ell)$ *with error $1 - 2^{-q}$ and* $\mathrm{poly}(q)$ *samples, where $\rho = 10\sqrt{d(n)/q(n)}$.*

4 A targeted HSG via bootstrapping systems

The main technical result underlying Theorem 1.2 is a construction of a reconstructive targeted HSG that is based on any function computable by logspace-uniform circuits of bounded depth. In this section we present this construction.

First, in Section 4.1 we formally define bootstrapping systems and discuss the complexity of functions with bootstrapping systems. Then, in Section 4.2 we state the two technical results that we will need for our HSG construction, and then use them to state the construction and prove it. Finally, in Sections 4.3 and 4.4 we prove each of the two technical results, respectively.

Throughout the section, we will frequently refer to integers in $[T]$ as representing gates in a given circuit of size T. This representation refers to the indices of gates according to an ordering in some fixed canonical way of describing the entire circuit as a list of gates.[19] We also consider integers that represent gates in individual layers of the circuit; this refers to indices of gates between $1, ..., T$ where T is the number of gates in the layer, where the ordering of gates is induced by the same global ordering of all the gates in the circuit.

4.1 Bootstrapping systems

The following definition of bootstrapping systems expands on the one that was presented in Section 2.2. The main difference is that in the definition below we distinguish the complexity of the algorithms for each of the different tasks associated with a bootstrapping system, whereas in Section 2.2 we just bounded them all by a single parameter (denoted t there). After the definition we demonstrate a typical setting of parameters that we will use.

Definition 4.1 (bootstrapping systems)**.** *Let $T, d, A\colon \mathbb{N} \to \mathbb{N}$ and $\rho\colon \mathbb{N} \to (0,1)$ be logspace-computable functions. We say that a function $f\colon \{0,1\}^* \to \{0,1\}^*$ has $(d \times T)$-*bootstrapping systems *with* alphabet size *A if for every $x \in \{0,1\}^n$ there exists a sequence of strings $P_0(x), ..., P_{d(n)}(x) \in [A(n)]^{T(n)}$ with the following properties:*

1. *(**Layers are efficiently printable.**) There exists an efficient algorithm that gets input $(x, i) \in \{0,1\}^n \times [d(n)]$ and prints the string $P_i(x)$ (using the natural encoding of integers in $[A(n)]$ as binary strings).*

2. *(**Base case.**) There exists an efficient algorithm that gets input $(x, j) \in \{0,1\}^n \times [T(n)]$ and outputs $P_0(x)_j$.*

3. *(**Downward self-reducibility.**) There exists an efficient algorithm that gets input $(x, i, j) \in \{0,1\}^n \times [d(n)] \times [T(n)]$ and oracle access to $P_{i-1}(x)$, and outputs $P_i(x)_j$.*

4. *(**Worst-case to rare-case reducibility.**) Each $P_i(x)$, considered as the truth-table of a function, is sample-aided worst-case to $\rho(n)$-rare-case self-reducible.*[20]

5. *(**Final case.**) There exists an efficient algorithm that gets input $x \in \{0,1\}^n$ and oracle access to $P_{d(n)}(x)$ and outputs $f(x)$.*

When claiming that a function f has bootstrapping systems, we will specify the precise complexities *of each of the efficient algorithms mentioned in Items (1), (2), (3), (4) and (5). We also call the parameter A the* alphabet size*, and the parameter ρ the* rare-case agreement*.*

[19]The only exception to this rule is in the proof of Claim 4.7.1, where a particular algorithm will need to describe circuits in a specific different way. We will explicitly mention this point in that proof.

[20]A minor technicality is that we defined sample-aided worst-case to rare-case reductions for Boolean functions $f\colon \{0,1\}^* \to \{0,1\}^*$, whereas here for each $n \in \mathbb{N}$ we have a collection of functions $\{P_i\}$ and we define the reduction for each function P_i in the collection. Our intention is that there exists a single uniform algorithm that gets input n and i and performs the reduction for P_i.

We will typically use bootstrapping systems in which the algorithm for printing layers runs in time $\mathrm{poly}(T)$, the algorithms in Items (2), (3), (4) and (5) run in time $t \ll T$ (say, $t = T^{\delta}$ for a very small constant $\delta > 0$), and the rare-case agreement is $\rho(n) = t^{-\Omega(1)}$. We note that there are several natural relaxations of our requirements from bootstrapping systems that still allow our main argument to follow through; see details below.

The complexity of functions with bootstrapping systems. Let f be a function that has a $(d \times T)$-bootstrapping with alphabet size $o(2^n)$, and assume that the algorithms for Items (2), (3), (4), and (5) all work in time t. We observe that f can be computable by logspace-uniform circuits of depth approximately $d \cdot t^2$ and size approximately $T \cdot (d \cdot t^2)$. In particular, for our parameter setting $\mathrm{poly}(d,t) \ll T$, these are logspace-uniform circuits of size T and depth $\mathrm{poly}(d,t) \ll T$.

Proposition 4.2 (functions with bootstrapping systems are computable by bounded-depth circuits)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be a function that has $(d \times T)$-bootstrapping systems with alphabet size $2^{o(n)}$, and assume that the algorithms for Items (2), (3), and (5) all work in time $t(n)$. Then f can be computed by a logspace-uniform circuit of depth $\tilde{O}(t(n)) \cdot d(n)$ and size $\tilde{O}(T(n) \cdot t(n)) \cdot d(n) \cdot n$.*

Proof. We rely on the fact that a time-t Turing machine can be simulated by a logspace-uniform circuit of size $O(t^2)$ (i.e., the standard method of computing the tableau of the machine's computation can be implemented by logspace-uniform circuits).

Given $x \in \{0,1\}^n$, iteratively for $i = 0, ..., d(n)$, the circuit computes a binary representation of $P_i(x)$, which is of length $T(n) \cdot o(n) \leq T(n) \cdot n$. By the base case in Definition 4.1, each block of $\log(o(n))$ bits in $P_0(x)$ can be computed in time t, and hence by a logspace-uniform circuit of size $O(t^2)$. Thus, we can compute the binary representation of $P_i(x)$ by a circuit of size $T(n) \cdot \tilde{O}(t(n))$. Similarly for $i \in [d(n)]$, we compute a binary representation of $P_{i+1}(x)$ from the precomputed binary representation of $P_i(x)$ using a logspace-uniform circuit of size $O(T(n) \cdot t(n)^2)$. Oracle gates of the circuit (which issue queries to the preceding layer $P_i(x)$) can be replaced by a gadget that implements the indexing function, which is computable by a logspace-uniform circuit of size $\tilde{O}(T(n))$. The output layer of the circuit uses the final case algorithm in a similar manner. The overall depth of this circuit is $(d(n) + 2) \cdot O(t(n)^2)$, and its size is $O(t(n)^2 \cdot T(n) \cdot d(n) \cdot n)$. ■

Relaxing the requirements. There are two natural relaxations of the requirements in Definition 4.1 that still allow our main argument (i.e., the construction of an HSG from bootstrapping systems in Proposition 4.4) to follow through. First, the requirement that each layer is worst-case to rare-case reducible can be relaxed, only requiring that each layer is a codeword in a code that has an efficient local decoder from a constant (say, 1/4) fraction of errors. This is the case because given a bootstrapping system that only satisfied the relaxed requirement, we can apply the derandomized direct product construction of Impagliazzo and Wigderson [IW99] to each layer: Their construction transforms each such codeword into the truth-table of a function that is sample-aided worst-case to rare-case reducible, where each entry in the new truth-table can be efficiently computed using logarithmically many queries to the previous truth-table (see Theorem A.5 for precise details).

Secondly, when presenting Definition 4.1 we implicitly assumed that the algorithms in Items (2), (3) and (5) are deterministic. While this is the case in our particular constructions, the proof of

Proposition 4.4 follows through even if these algorithms are probaiblistic (this is since the reconstruction argument that uses these algorithms is probabilistic to begin with).

Previous works as using bootstrapping systems. As mentioned in Section 2, previous works in uniform hardness-to-randomness can be viewed in retrospect as relying on bootstrapping systems. Specifically, in the works of Impagliazzo and Wigderson [IW98] and of Trevisan and Vadhan [TV07] the hard function was a set $L \subseteq \{0,1\}^*$ that is downward self-reducible and randomly self-reducible (instead of a multi-output function f as in our work), and given $x \in \{0,1\}^n$ considered the following bootstrapping system: For each $i \in [n]$, the layer P_i is the truth-table of L on inputs of length i.

To see that this yields bootstrapping systems as in Definition 4.1, note the downward self-reducibility of L yields the algorithm in Item (3), and the random self-reducibility of L ensures that each row in the bootstrapping system is a codeword in a code that has an efficient local decoder (which, as explained above, suffices to meet the requirement in Item (4)). The layers in their constructions are printable in polynomial space, and the algorithms for Items (2) and (5) follow since L_1 is trivially-computable and since L_n at location x is simply the sought value $L(x)$.

An important point to notice, which we already mentioned in Section 2, is that the foregoing bootstrapping systems *are identical for all inputs of a given length n* (i.e., the bootstrapping system matrices $B_L(x)$ and $B_L(x')$ are identical for every $x, x' \in \{0,1\}^n$). In contrast, our bootstrapping systems will depend on the particular input.

4.2 The result statements: A reconstructive targeted HSG

We now state the two main technical results that we need for our HSG construction, and then combine them to obtain the HSG construction. The first technical result asserts that any function computable by logspace-uniform circuits of bounded depth has efficient bootstrapping systems. When parsing the parameters below, we encourage the reader to notice that the algorithms for computing P_0, for downward self-reducibility, and for worst-case to rare-case reducibility, all run in time much smaller than T (i.e., in time $t = T^{\mu}$ for a very small μ).

Proposition 4.3 (bootstrapping systems based on the proof system of [GKR15])**.** *There exist two universal constants $\alpha \in (0,1)$ and $k > 1$ such that the following holds. Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be length-preserving function computable by a logspace-uniform family $\{C_n\}_{n\in\mathbb{N}}$ of circuits of size at most $T(n)^k$ and depth $d(n)$. Then, there exist $d' = O(d \cdot \log(T))$ and $T' = \text{poly}(T)$ such that for every logspace-computable $\mu\colon \mathbb{N} \to (0,1)$ satisfying $t = T^{\mu} \geq \log^{1/\alpha}(T)$, the function f has $(d' \times T')$-bootstrapping systems with alphabet size $O(\log(T))$ and the following properties:*

1. *The algorithm that prints $P_i(x)$ is a logspace-uniform circuit of size $\text{poly}(T)$ and depth $d' + O(\log^2(T))$.*

2. *The algorithm that computes $P_0(x)$ is a logspace-uniform circuit of size $\max\{n,t\} \cdot t$ and depth $O(\log^2(T))$.*

3. *The downward self-reducibility algorithm is a logspace-uniform circuit of size t and depth $O(\log^2(T))$.*

4. *The sample-aided worst-case to ρ-rare-case algorithm supports agreement $\rho = t^{-\alpha} \cdot \mathrm{polylog}(T)$, has error $2^{-t^{\alpha}}$, and is computable by logspace-uniform circuits of size t and depth $\log^{1/\alpha}(T)$.* [21]

The second technical result below constructs a reconstructive targeted HSG based on any function f that has an efficient bootstrapping system. We stress that the transformation does *not depend on f being logspace-uniform or having bounded-depth circuits*, but holds for any function with bootstrapping systems. (Indeed, only in the "moreover" we assume that f has this particular form, in which case we deduce that the algorithms associated with the HSG are also logspace-uniform circuits of bounded depth.) When parsing the parameters below, we encourage the reader to think of t and t_0 as very small compared to T' (e.g., $t = T^{\gamma}$ for a very small γ).

Proposition 4.4 (from bootstrapping systems to a targeted HSG)**.** *There exist universal constants $c', c'' > 1$ such that the following holds.*

Assumption: For $T', A, \bar{T}, t_0, t, d' \colon \mathbb{N} \to \mathbb{N}$ and $\alpha \in (0,1)$ such that $\max\{A(n), t(n), t_0(n), d'(n)\} \leq T'(n)$, let f be a length-preserving function that has $(d' \times T')$-bootstrapping systems with alphabet size A satisfying the following:

1. *The algorithm that prints $P_i(x)$ runs in time $\bar{T}$.*
2. *The algorithm that computes $P_0(x)$ runs in time t_0.*
3. *The downward self-reduction runs in time t.*
4. *The sample-aided worst-case to rare-case reduction runs in time t, supports a rare-case agreement of $t^{-\alpha}$, and has error $2^{-t^{\alpha}}$.*

Conclusion: Then, for every time-computable $M \colon \mathbb{N} \to \mathbb{N}$ and $\gamma \colon \mathbb{N} \to (0,1)$ such that $\log(T') \leq M \leq \min\left\{(T')^{\gamma/c''}, t^{\alpha/c''}\right\}$ there exist a deterministic algorithm H_f and a probabilistic algorithm R that for every $x \in \{0,1\}^n$ satisfy the following:

1. **Generator.** *When H_f gets input $x \in \{0,1\}^n$ it runs in time $(\bar{T} + (T')^{1/\gamma})^{c'}$ and outputs a set of M-bit strings.*
2. **Reconstruction.** *When R gets input x and oracle access to a function $D \colon \{0,1\}^M \to \{0,1\}$ such that $\Pr_{r \in \{0,1\}^M}[D(r) = 1] \geq 1/M$ but D rejects all the strings that $H_f(x)$ outputs, it runs in time $(t \cdot (T')^{\gamma} \cdot M)^{c'} \cdot \left(d' + n + t_0^{c'}\right)$ and with probability at least $1 - d' \cdot \left(1/(T')^2 + 3 \cdot 2^{-M}\right)$ outputs $f(x)$.*

Moreover, assume that γ is constant, that all the relevant functions (i.e., $T', A, \bar{T}, t_0, t, d'$, and M) are logspace-computable, that the algorithms in Items (1), (2), (3), and (4) of the hypothesis are logspace-uniform circuits of size identical to the stated time bounds, that the circuit in Item (1) has depth $\widetilde{d}$, and that the circuits in the other three items have depth $\mathrm{polylog}(T')$. Then, generator H_f can be computed by a logspace-uniform circuit of size $\mathrm{poly}(T')$ and depth $\widetilde{d} + \mathrm{polylog}(T')$, and the reconstruction algorithm R can be computed by a logspace-uniform probabilistic circuit of size $(t \cdot (T')^{\gamma} \cdot M)^{c'} \cdot \left(d' + n + t_0^{c'}\right)$ and depth $d' \cdot \mathrm{polylog}(T')$.

[21] The polylogarithmic power in the definition of ρ depends on the family $\{C_n\}$.

As mentioned above, the proofs of Propositions 4.3 and 4.4 appear in Sections 4.3 and 4.4, respectively. The following result is our main construction of the HSG in this section, and its proof amounts to a straightforward combination of the foregoing two results. We state the result for a relatively high reconstruction overhead (i.e., T^{δ} for a constant $\delta > 0$), in which case the HSG and reconstruction are guaranteed to be a logspace-uniform circuit of bounded depth; a statement allowing lower overheads (i.e., $T^{o(1)}$), but without such guarantee, appears in Section 5.

Proposition 4.5 (a reconstructive targeted HSG)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $f\colon \{0,1\}^n \to \{0,1\}^n$ be computable by logspace-uniform circuits of size $T(n)$ and depth $d(n)$, let $\delta > 0$, and let $M\colon \mathbb{N} \to \mathbb{N}$ such that $c \cdot \log(T(n)) \le M(n) \le T(n)^{\delta/c}$. Then, there exist a deterministic algorithm H_f and a probabilistic algorithm R that for every $x \in \{0,1\}^n$ satisfy the following:*

1. **Generator.** *The generator H_f gets input x and outputs a set of M-bit strings. It is computable by a logspace-uniform circuit of size $\mathrm{poly}(T)$ and depth $(d + \log(T)) \cdot O(\log(T))$.*

2. **Reconstruction.** *When R gets input x and oracle access to a function $D\colon \{0,1\}^M \to \{0,1\}$ such that $\Pr_{r \in \{0,1\}^M}[D(r) = 1] \ge 1/M$ but D rejects all the strings that $H_f(x)$ prints, it outputs $f(x)$ with probability at least $1 - 1/M$. The procedure R is computable by a logspace-uniform probabilistic circuit of size $(d + n^c) \cdot T^{\delta} \cdot M^c$ and depth $d \cdot \log^c(T)$.*

Proof. Let c' and c'' be the universal constants from Proposition 4.4, and let $\alpha \in (0,1)$ and k be the universal constants from Proposition 4.3. We instantiate the bootstrapping systems for f from Proposition 4.3 with parameter $\mu = \delta/5c'$. This yields a system with dimensions $(d' \times T')$ for $d' = O(d \cdot \log(T))$ and $T' = T^k$, where the algorithm that prints each P_i is a logspace-uniform circuit of size $\bar{T} = T^{O(1/\mu)} = \mathrm{poly}(T)$ and depth $d' + O(\log^2(T))$, the algorithms computing the two reductions (i.e., downward self-reducibility and worst-case to rare-case reducibility) are logspace-uniform circuits of size $t = T^{\delta/5c'}$ and depth $\mathrm{polylog}(T)$, and the algorithm that computes P_0 is a logspace-uniform circuit of size $t_0 = \max\{n, t\} \cdot t$ and depth $\mathrm{polylog}(T)$.

We now plug this system into Proposition 4.4, using the parameter $\gamma = \delta/(5k \cdot c')$. The hypothesis in Proposition 4.4 that $A(n) \le T(n)$ is satisfied (since the bootstrapping system has alphabet size $A(n) = O(\log(T(n)))$), and the constraint in its conclusion that $k \cdot \log(T) \le M \le \min\left\{t^{\alpha/c''}, (T')^{\gamma/c''}\right\} = \min\left\{T^{(\alpha\cdot\delta)/(5c'\cdot c'')}, T^{\delta/(5c'\cdot c'')}\right\} = T^{(\alpha\cdot\delta)/(5c'\cdot c'')}$ is satisfied by our hypothesis that $c \cdot \log(T) \le M \le T^{\delta/c}$ for a sufficiently large universal constant $c \ge \max\{k, (5c' \cdot c'')/\alpha\}$.

Note that the generator H_f is indeed computable by a logspace-uniform circuit of size $\mathrm{poly}(\bar{T} + (T')^{1/\gamma}) = \mathrm{poly}(T)$ and depth

$$\widetilde{d} + O(\log^2(T)) = d' + O(\log^2(T)) = O(d \cdot \log(T)) + O(\log^2(T)) \ .$$

The reconstruction algorithm R can be computed by a logspace-uniform probabilistic circuit of size

$$\left(t \cdot T^{k\gamma} \cdot M\right)^{c'} \cdot \left(d' + n + (\max\{n,t\} \cdot t)^{c'}\right) < T^{\delta} \cdot M^{c} \cdot (n^{c} + d)$$

and of depth $d' \cdot \mathrm{polylog}(T') = d \cdot \log^c(T)$, assuming that the universal constant c is sufficiently large. The probability that R errs is at most

$$O(d \cdot \log(T)) \cdot \left(\frac{1}{(T')^2} + 3 \cdot 2^{-M} \right) < 1/M \,. \quad \blacksquare$$

4.3 Bootstrapping systems for logspace-uniform bounded-depth circuits

In this section we prove Proposition 4.3. As explained in Section 2, our construction mimics the interactive proof system of Goldwasser, Kalai, and Rothblum [GKR15]. Specifically, we arithmetize the circuit layer-by-layer over an appropriate arithmetic setting, and "in between layers" we add additional polynomials that represent a suitable sumcheck protocol, which allows reducing claims about each layer to claims about the preceding layer. Our construction is actually simpler, and does not refer to two auxiliary interactive protocols from their original proof system; this simplification is possible since we only need a bootstrapping system rather than a proof system, and using an additional idea of Goldreich [Gol18].[22]

We first define the notion of a polynomial decomposition of a circuit, then show that any function computable by logspace-uniform bounded-depth circuits has a polynomial decomposition with good parameters, and finally show how a function with such a polynomial decomposition has bootstrapping systems with good parameters.

Definition 4.6 (polynomial decomposition of a circuit)**.** *Let C be a circuit that has n input bits, fan-in two, size T, and depth d. For every $x \in \{0,1\}^n$, we call a collection of polynomials a* polynomial decomposition of $C(x)$ *if it meets the following specifications:*

1. **(Arithmetic setting.)** *For some prime $p \leq T$, the polynomials are defined over the prime field $\mathbb{F} = \mathbb{F}_p$. For some integer $h \leq p$, let $H = [h] \subseteq \mathbb{F}$, let m be the minimal integer such that $h^m \geq T$, and let $m' \leq m$ be the minimal integer such that $h^{m'} \geq n$.*

2. **(Circuit-structure polynomial.)** *For each $i \in [d]$, let $\Phi_i \colon H^{3m} \to \{0,1\}$ be the function such that $\Phi_i(\vec{w}, \vec{u}, \vec{v}) = 1$ if and only if the gate in layer i indexed by $\vec{w}$ is fed by the gates in layer $i-1$ indexed by $\vec{u}$ and $\vec{v}$. (If one of the elements $\vec{w}, \vec{u}, \vec{v}$ does not index a valid gate in the corresponding layer, then Φ_i outputs zero.) The polynomial $\hat{\Phi}_i \colon \mathbb{F}^{3m} \to \mathbb{F}$ can be any extension of Φ_i.*

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

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

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

[22]Let us spell out the differences, for readers who are familiar with [GKR15]. First, following the simplification idea of Goldreich [Gol18], we avoid an auxiliary protocol from their original system intended to compute the circuit-structure polynomial. Secondly, we avoid an auxiliary protocol that reduces verification of a pair of points at each step to verification of a single point at each step; this is because in bootstrapping systems we are fine with verifying a set of points at each step (in contrast to proof systems, where this yields an exponential blow-up in the verification time). Lastly, we warn readers that our indexing of the layers below is reverse order compared to the indexing in their paper (i.e., we index from bottom to top rather than from top to bottom).

4. **(Layer polynomials.)** *For each $i \in [d]$, let $\alpha_i \colon H^m \to \{0,1\}$ represent the values of the gates at the i^{th} layer of C in the computation of $C(x)$ (with zeroes in locations that do not index valid gates), and let $\hat{\alpha}_i \colon \mathbb{F}^m \to \mathbb{F}$ be defined by*

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

5. **(Sumcheck polynomials.)** *For each $i \in [d]$, let $\hat{\alpha}_{i,0} \colon \mathbb{F}^{3m} \to \mathbb{F}$ be the polynomial*

$$\hat{\alpha}_{i,0}(\vec{w},\sigma_1,...,\sigma_{2m}) = \hat{\Phi}_i(\vec{w},\sigma_{1,...,m},\sigma_{m+1,...,2m}) \cdot (1 - \hat{\alpha}_{i-1}(\sigma_{1,...,m}) \cdot \hat{\alpha}_{i-1}(\sigma_{m+1,...,2m})) \ ,$$

and for every $j \in [2m-1]$, let $\hat{\alpha}_{i,j} \colon \mathbb{F}^{3m-j} \to \mathbb{F}$ be the polynomial

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

where $\sigma_{k,...,k+r} = \sigma_k, \sigma_{k+1}, ..., \sigma_{k+r}$. Lastly, let $\hat{\alpha}_{i,2m}(\vec{w}) = \hat{\alpha}_i(\vec{w})$.

We now show that any function computable by logspace-uniform circuits of bounded depth has a polynomial decomposition whose reducibility algorithms are very efficient.

Proposition 4.7 (polynomial decompositions of logspace-uniform circuits using universal circuits). *There exist two universal constants $c, c' \in \mathbb{N}$ such that the following holds. Let $\{C_n\}_{n \in \mathbb{N}}$ be a logspace-uniform family of circuits of size $T(n)$ and depth $d(n)$, and let $\gamma \colon \mathbb{N} \to (0,1)$ be a logspace-computable function such that $T(n)^{\gamma(n)} \geq \log(T(n))$. Then, there exists a logspace-uniform family of circuits $\{C'_n\}_{n \in \mathbb{N}}$ of size $T'(n) = O(T(n)^c)$ and depth $d'(n) = O(d(n) \cdot \log(T(n)))$ that computes the same Boolean function as $\{C_n\}$ such that for every $x \in \{0,1\}^n$ there exists a polynomial decomposition of $C'_n(x)$ satisfying:*

1. **(Arithmetic setting.)** *The polynomials are defined over $\mathbb{F}_p$, where p is the smallest prime in the interval $[T^{\gamma \cdot c}, 2T^{\gamma \cdot c}]$. Let $H = [h] \subseteq \mathbb{F}$, where h is the smallest power of two of magnitude at least $T^{\gamma/6}$, and let m be the minimal integer such that $h^m \geq 2T^c$.*

2. **(Faithful representation.)** *For every $i \in [d'(n)]$ and $\vec{w} \in H^m$ representing a gate in the i^{th} layer it holds that $\hat{\alpha}_i(\vec{w})$ is the value of the gate $\vec{w}$ in $C'_n(x)$.*

3. **(Layer downward self-reducibility.)** *There is a logspace-uniform oracle circuit of size* $\max\{n,h\} \cdot h^{c'}$ *and depth $O(\log^2(T))$ that, when given oracle access to $x \in \{0,1\}^n$, computes the function $\hat{\alpha}_0$. Also, there is a logspace-uniform oracle circuit of size $h^{c'}$ and depth $O(\log^2(T))$ that computes $\hat{\alpha}_{i,0}$ while querying $\hat{\alpha}_{i-1}$ twice on inputs in H^m. Lastly, $\hat{\alpha}_{i,2m} \equiv \hat{\alpha}_i$.*

4. **(Sumcheck downward self-reducibility.)** *There is a logspace-uniform oracle circuit of size $h^{c'}$ and depth $O(\log(T))$ that gets input $\vec{w} \in \mathbb{F}^m$ and $(\sigma_1,...,\sigma_{2m}) \in \mathbb{F}^{2m}$ and $j \in [2m]$ and oracle access to $\hat{\alpha}_{i,j-1}$ and outputs $\hat{\alpha}_{i,j}(\vec{w},\sigma_1,...,\sigma_{2m-j})$.*

5. **(Sample-aided worst-case to rare-case reducibility.)** *For each $i \in [d'(n)]$ and $j \in [2m]$, the Boolean function representing $\hat{\alpha}_{i,j}$ is sample-aided worst-case to ρ-rare-case reducible with error 2^{-h}*

by logspace-uniform circuits of size $h^{c'}$ and depth $\mathrm{polylog}(T)$*, where* $\rho = h^{-c} \cdot \mathrm{polylog}(T)$*. The same claim holds for* $\hat{\alpha}_0$*.*

Proof. Following [Gol18], we consider a circuit C'_n that first computes a description of C_n (represented as a $T(n) \times T(n) \times T(n)$ tensor) and then computes the `Eval` function $(\langle C_n \rangle, x) \mapsto C_n(x)$. The construction of C'_n in [Gol18] essentially suffices for our purposes, but we will need to use a property of this construction that was not explicitly stated in [Gol18] (specifically, the fact that the adjacency relation of C'_n is computable in small space). That is:

Claim 4.7.1. *There exists a circuit C'_n as above of depth $d'(n) = O(\log^2(T(n)) + d(n) \cdot \log(T(n)))$ and size $T' = 2^{c \cdot \lceil \log(T(n)) \rceil}$ (for some universal integer $c > 1$) such that the layered adjacency relation function $\Phi' \colon [d'] \times \{0,1\}^{3\log(T')} \to \{0,1\}$ of C'_n can be decided by a formula that can be constructed in time* $\mathrm{polylog}(T(n))$ *and space* $O(\log(T(n)))$*.*

Proof. We follow the construction of C'_n and its analysis in [Gol18, Sections 3.3 and 3.4.2], while tracking the relevant changes. The original construction is stated with respect to a parameter n such that the input length is n and C_n is a circuit of size $\mathrm{poly}(n)$ and depth $d(n)$ whose adjacency relation can be decided in time $O(\log(n))$. However, this construction scales smoothly to the case where C_n is of size $T(n)$ and depth $d(n)$ and the adjacency relation can be decided in time $O(\log(T(n)))$, by considering C_n as a circuit over $T(n)$ bits that ignores all but the first n bits. Also, without loss of generality, we can assume that $T'(n)$ is a power of two.

Turning to the adjacency relation Φ', in [Gol18] it is only stated that Φ' can be computed by a formula that can be constructed in time $\mathrm{polylog}(T(n))$. To see that the algorithm constructing the formula only uses space $O(\log(T(n)))$, note that each of the three stages of the construction of C'_n yields a very simple description of the adjacency relations in the corresponding part of C'_n: The first step consists of constructing the matrix of transitions between instantaneous configurations of the $O(\log(T))$-space machine that prints a description of C_n (the gates in this step have no incoming wires); the second steps consists of squaring the foregoing matrix for $O(\log(T(n)))$ times; and the third step consists of computing the `Eval` function with input $(\langle C_n \rangle, x)$. (See [Gol18, Section 3.4.2] for explicit descriptions of the adjacency relations in the two latter parts.) □

Since $d' \leq T'$, we can extend Φ' to a function $\{0,1\}^{4\log(T')} \to \{0,1\}$ without noticeably affecting its complexity. We denote the size of the formula for Φ by $s' = \mathrm{polylog}(T)$, and for every $i \in [d']$, we denote by $\Phi_i(\cdot) = \Phi'(i, \cdot)$ the i^{th} "slice" of Φ' (that is, $\Phi_i(\vec{w}, \vec{u}, \vec{v}) = \Phi'(i, \vec{w}, \vec{u}, \vec{v})$.

Recall that p is the smallest prime in the interval $[T^{\gamma \cdot c}, 2T^{\gamma \cdot c}]$, and that h is the smallest power of two of magnitude at least $T^{\gamma/6}$. Note that given input 1^n, both p and h can be found in space $O(\log(T))$. For every $x \in \{0,1\}^n$, we consider the polynomial decomposition of $C'_n(x)$ with $h \leq p \leq T$. The claim about faithful representation holds for any valid arithmetic extension of the Φ_i's. Thus, to complete the description of the decomposition and conclude the proof, we now specify the arithmetization $\hat{\Phi}_i$ of each Φ_i, and then prove our claims regarding downward self-reducibility and worst-case to rare-case reducibility.

Arithmetization of the Φ_i's. Our goal now is to arithmetize each Φ_i as a polynomial $\mathbb{F}^{3m} \to \mathbb{F}$ that has low degree and is efficiently computable, as follows:

Claim 4.7.2. *For $i \in [d']$ there exists $\hat{\Phi}_i \colon \mathbb{F}^{3m} \to \mathbb{F}$ that satisfies the following:*

1. *For every* $(\vec{w}, \vec{u}, \vec{v}) = z_1, ..., z_{3m} \in H^{3m}$ *we have that* $\hat{\Phi}_i(\vec{w}, \vec{u}, \vec{v}) = 1$ *if gate* $\vec{w}$ *in the* i^{th} *layer of* C'_n *is fed by gates* $\vec{u}$ *and* $\vec{v}$ *in the* $(i-1)^{th}$ *layer of* C'_n*, and* $\hat{\Phi}_i(\vec{w}, \vec{u}, \vec{v}) = 0$ *otherwise.*

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

3. *For a universal constant* $c_1 > 1$*, there exists a logspace-uniform circuit of size* h^{c_1} *and depth* $O(\log^2(T))$ *that computes* $\hat{\Phi}_i$*.*

Proof. We think of Φ_i as a function $\mathbb{F}_2^{3\log(T')} \to \mathbb{F}_2$, and note that it is computable by an *arithmetic formula* (over $\mathbb{F}_2$), whose structure mimics the one of the original Boolean formula (i.e., each gate w in the original formula computes the NAND of two sub-formulas u and v, and thus w can be replaced by the expression $\hat{w} = 1 - \hat{u} \cdot \hat{v}$). The same arithmetic formula can be used to compute a polynomial $\Phi'_i \colon \mathbb{F}^{3\log(T')} \to \mathbb{F}$ of degree $\mathrm{poly}(s')$ that agrees with Φ_i on $\mathbb{F}_2^{3\log(T')}$. [23]

Next, let $\ell = \log(h)$, and for every $j \in [\ell]$ consider the function $\pi_j \colon H \to \{0,1\}$ such that $\pi_j(a)$ is the j^{th} bit in the binary representation of the integer a. Note that there is a polynomial $\hat{\pi}_j \colon \mathbb{F} \to \mathbb{F}$ of degree at most h that agrees with π_j on H. Finally, let $\hat{\Phi}_i \colon \mathbb{F}^{3m} \to \mathbb{F}$ such that

$$\hat{\Phi}_i(z_1, ..., z_{3m}) = \Phi'_i(\hat{\pi}_1(z_1), ..., \hat{\pi}_\ell(z_1), ..., \hat{\pi}_1(z_{3m}), ..., \hat{\pi}_\ell(z_{3m})) \ . \tag{4.1}$$

By definition, when $\hat{\Phi}_i$ is given input $(\vec{w}, \vec{u}, \vec{v}) \in H^{3m}$, the π_j's project each of the three inputs to a bit-string that is the index of a gate, and the arithmetic formula Φ'_i computes Φ_i on the corresponding gates. Also, the degree $\Delta \stackrel{\mathrm{def}}{=} \deg(\hat{\Phi}_i)$ satisfies $\Delta \le h \cdot \mathrm{poly}(s') = h \cdot \mathrm{polylog}(T)$.

Lastly, the algorithm for $\hat{\Phi}_i$ gets input $z_1, ..., z_{3m}$, computes $\hat{\pi}_j(w_k)$ for each $k \in [3m]$ and $j \in [\ell]$, and evaluates the arithmetic formula for Φ'_i on the resulting sequence of $3m \cdot \ell$ elements from $\mathbb{F}$. Now, recall that addition of h elements in $\mathbb{F}_p$ and iterated multiplication of h elements in $\mathbb{F}_p$ are computable by logspace-uniform circuits of depth $O(\log(p))$ and size $\mathrm{poly}(h \cdot \log(p)) = \mathrm{poly}(h)$ (see Lemma B.1)). Thus, using the formula $\hat{\pi}_j(u) = \sum_{a \in H} \pi_j(a) \cdot \prod_{a' \in H \setminus \{a\}} \frac{u-a'}{a-a'}$, the polynomial $\hat{\pi}_j$ can be computed by a logspace-uniform circuit of depth $O(\log(p)) = O(\log(T))$ and size $\mathrm{poly}(h)$. Also, since the formula Φ_i is logspace-uniform and is of size $\mathrm{polylog}(T)$ and depth $O(\log(T))$, we can replace each gate in Φ_i by a logspace-uniform circuit of depth $O(\log(p)) = O(\log(T))$ and size $\mathrm{polylog}(p) = \mathrm{polylog}(T)$ for the corresponding arithmetic operation and obtain a logspace-uniform circuit of size $\mathrm{polylog}(T)$ and depth $O(\log^2(T))$ for the arithmetic version of Φ_i. The final algorithm for $\hat{\Phi}_i$ is thus a logspace-uniform circuit of size

$$\ell \cdot m \cdot \mathrm{poly}(h) + \mathrm{polylog}(T) \le h^{c_1} \ ,$$

and depth $O(\log^2(T))$, where we relied on the fact that $m \le \log(T) \le \mathrm{poly}(h)$ and that $c_1 > 1$ is a universal constant (which depends on the size of the logspace-uniform circuits for iterated multiplication). □

Layer downward self-reducibility. Recall that, by Definition 4.6, m' is the minimal integer such that $h^{m'} \ge n$, and that $\delta_{\vec{z}}$ is Kronecker's delta function. By the definition of $\hat{\alpha}_0$, to compute it we can first enumerate (in parallel) over all elements in $H^{m'}$, then enumerate over $j \in [m]$ in parallell;

[23]Note that the domain size of Φ'_i is superpolynomial in T, but none of our algorithms will explicitly construct the entire truth-table of Φ'_i at any point.

for each element and j perform $(h-1)$ multiplication operations on our input, and then multiply the m results; and finally sum up the $h^{m'}$ results. Relying on Lemma B.1, this can be done by logspace-uniform circuits of size

$$h^{m'} \cdot m \cdot \text{poly}(h) + O(h^{m'} \cdot \log(p \cdot h^{m'})) \leq \max\{n, h\} \cdot h^{c_2}$$

and of depth $O(\log(p)) + O(\log(p) \cdot \log(n)) = O(\log^2(T))$, for some universal constant $c_2 > 1$. (We again relied on the fact that $m \leq \log(T) \leq \text{poly}(h)$.)

To compute $\hat{\alpha}_{i,0}$ with an oracle to $\hat{\alpha}_{i-1}$, we get inputs $(\vec{w}, \vec{u}, \vec{v})$ and we need to compute $\hat{\Phi}_i(\vec{w}, \vec{u}, \vec{v})$ and perform two oracle calls and arithmetic operations; by Claim 4.7.2, we can do so with a logspace-uniform circuit of size $h^{c_1} + O(\log(p))$ and depth $O(\log^2(T))$. The identity $\hat{\alpha}_{i,2m} \equiv \hat{\alpha}_i$ is by definition.

Sumcheck downward self-reducibility. The algorithm for $\hat{\alpha}_{i,j}$ follows from the fact that

$$\hat{\alpha}_{i,j}(\vec{w}, \sigma_1, ..., \sigma_{2m-j}) = \sum_{\sigma_{2m-j+1} \in H} \hat{\alpha}_{i,j-1}(\vec{w}, \sigma_1, ..., \sigma_{2m-j+1}) \,,$$

where the RHS can be computed by adding the answers to h oracle queries to $\hat{\alpha}_{i,j-1}$. We construct a uniform circuit that takes input j and $\vec{w}, \sigma_1, ..., \sigma_{2m}$, enumerates (in parallel) all $\sigma \in H$, uses each fixed σ along with the first $2m - j$ elements $\sigma_1, ..., \sigma_{2m-j}$ to issue an oracle query $\sigma_1, ..., \sigma_{2m}, \sigma$ to $\hat{\alpha}_{i,j-1}$, and adds the h results. This can be done by logspace-uniform circuits of size h^{c_3} and depth $O(\log(p)) = O(\log(T))$, where $c_3 > 1$ is a universal constant.

Worst-case to rare-case reducibility. We claim that for every $i \in [d']$, the degree of $\hat{\alpha}_i$ and of $\hat{\alpha}_{i,j}$, is at most $\Delta \overset{\text{def}}{=\!=} h \cdot \text{polylog}(T)$. To see this, for $i \in [d']$, note that $\hat{\alpha}_i$ feeds its input $\vec{w}$ only to the function $\hat{\Phi}_i$, and recall that $\deg(\hat{\Phi}_i) = h \cdot \text{polylog}(T) \leq \Delta$. An identical argument shows that for every $i \in [d']$ and $j \in [2m]$, the degree of $\hat{\alpha}_{i,j}$ is at most Δ. Now, note that

$$\Delta / |\mathbb{F}| \leq h \cdot \text{polylog}(T) / (T')^{\gamma} \leq h \cdot T^{-\gamma \cdot c} \cdot \text{polylog}(T) \,.$$

By Proposition 3.10, the Boolean function associated with $\hat{\alpha}_i$ is sample-aided worst-case to ρ-rare-case reducible, for

$$\rho = 10\sqrt{\Delta / |\mathbb{F}|} < T^{-(\gamma \cdot c)/2} \cdot \sqrt{h} \cdot \text{polylog}(T) < T^{-c\gamma/6} \cdot \text{polylog}(T) \,,$$

which is upper-bounded by $h^{-c} \cdot \text{polylog}(T)$. The reduction can be computed by logspace-uniform circuits of size $\text{poly}(m, T^{\gamma \cdot c}) \leq h^{c_4}$ using at most h^{c_4} samples (for some universal $c_4 \geq 1$) and depth $\text{polylog}(T)$ and has error $2^{-|\mathbb{F}|} < 2^{-h}$.

Wrapping-up. To conclude we define the constant c' to satisfy $c' \geq \max\{c_1 + 1, c_2, c_3, c_4\}$. ∎

We now prove Proposition 4.3, which is our construction of bootstrapping systems. The proof amounts to transforming the polynomial decomposition from Proposition 4.7 into bootstrapping systems, by ordering the layer polynomials and the sumcheck polynomials in an appropriate ascending order (and carefully verifying the parameters of the resulting construction).

Proof of Proposition 4.3. Let C_n be the logspace-uniform circuit for f on inputs of length n, which is of depth $d = d(n)$ and of size $T = T(n)$. Let c and c' be the universal constants from Proposition 4.7, and let C'_n be the corresponding logspace-uniform circuit from Proposition 4.7, which has depth $d'_0 = O(d \cdot \log(T))$ and size $T' = O(T)^c < T^k$ (for any constant $k > c$). For $\gamma = 5\mu/c'$, consider the corresponding polynomial decomposition of C'_n from Proposition 4.7; the requirement that $T^\gamma > \log(T)$ is satisfied by our hypothesis that $T^\mu \geq \log^{1/\alpha}(T)$ and by a choice of sufficiently small $\alpha \leq 1/c'$. Note that $|\mathbb{F}| \leq \mathrm{poly}(T)$ and $h = O(T^{5\mu/6c'})$ and $m = O(1/\mu)$. Denote $t = h^{c'} = O(T^{5\mu/6}) < T^\mu$.

Let $\{\hat{\alpha}_i : \mathbb{F}^m \to \mathbb{F}\}_{i \in \{0,\ldots,d'_0\}}$ and $\{\hat{\alpha}_{i,j} : \mathbb{F}^{3m-j} \to \mathbb{F}\}_{i \in [d'_0], j \in \{0,\ldots,2m-1\}}$ be the corresponding layer polynomials and sumcheck polynomials in the decomposition, respectively. We now view each $\hat{\alpha}_i$ and $\hat{\alpha}_{i,j}$ as a Boolean function mapping $3m \cdot \log(|\mathbb{F}|) = O(\log(T))$ bits to $|\mathbb{F}| = O(\log(T))$ bits (note that we think of all these functions as having the same domain, meaning that some of the Boolean functions will ignore a suffix of their input).

Defining the bootstrapping system. The bootstrapping system has $d' = d'_0 \cdot (2m+1) = O(d'_0/\mu)$ layers, each of length $|\mathbb{F}|^{3m} = \mathrm{poly}(T)$ and over alphabet $\mathbb{F}$, in addition to the base layer P_0. The base layer P_0 is the truth-table of the function $\hat{\alpha}_0$, and for each $(i,j) \in [d'_0] \times \{0,\ldots,2m\}$, the $(i,j)^{th}$ layer is the truth-table of the function $\hat{\alpha}_{i,j}$. The layers are ordered first according to an increasing value of i, and then according to an increasing order of j; for example, the following three layers are listed in ascending order: $(1, 2m-1)$, then $(1, 2m)$, and then $(2, 0)$.

Printing each layer. Given $x \in \{0,1\}^n$ and an index $(i,j) \in [d'_0] \times \{0,\ldots,2m\}$, we print the layer corresponding to (i,j) as follows. We first compute the values of all the gates of C'_n, which can be done by a logspace-uniform circuit of size $\mathrm{poly}(T)$ and depth d'_0. This gives us the values of $\hat{\alpha}_{i-1}$ on the set H^m (because these are the values of the gates in the $(i-1)^{th}$ layer of $C'_n(x)$, which in the case of $i = 1$ are just the bits of x). We compute the truth-table of $\hat{\alpha}_{i,0}$, using the layer self-reducibility algorithm and our oracle access to values of $\hat{\alpha}_{i-1}$ on H^m; this can be done by a logspace-uniform circuit of size $t \cdot \mathrm{poly}(T) = \mathrm{poly}(T)$ and depth $O(\log^2(T))$. Then, iteratively for $j' = 1, \ldots, j$, we compute the truth-table of $\hat{\alpha}_{i,j'}$, using the sumcheck self-reducibility algorithm and oracle access to $\hat{\alpha}_{i,j'-1}$; this can be done by a logspace-uniform circuit of size $O(m) \cdot t \cdot \mathrm{poly}(T) = \mathrm{poly}(T)$ and depth $O(m \cdot \log(T)) = O(\log(T)/\mu)$. Thus, the final algorithm that prints the $(i,j)^{th}$ layer is a logspace-uniform circuit of size $\mathrm{poly}(T)$ and depth $d'_0 + O(\log^2(T))$.

Base case. Given $x \in \{0,1\}^n$ and $k \in [\mathrm{poly}(T)]$, we can use the algorithm for $\hat{\alpha}_0$ and our access to x to output $P_0(x)_k$. This can be done by a logspace-uniform circuit of size $\max\{n, h\} \cdot t \leq \max\{n, t\} \cdot t$ and depth $O(\log^2(T))$.

Downward self-reducibility. We are given $x \in \{0,1\}^n$ and $(i,j) \in [d'_0] \times \{0,\ldots,2m\}$. The proof differs according to whether $j = 0$ or $j \in [2m]$:

1. If $j = 0$, then we need to compute $\hat{\alpha}_{i,0}$ with oracle access to $\hat{\alpha}_{i-1,2m} = \hat{\alpha}_{i-1}$. By layer self-reducibility, we can do so by a logspace-uniform circuit of size t and depth $O(\log^2(T))$.

2. If $j \in [2m]$, we need to compute $\hat{\alpha}_{i,j}$ with oracle access to $\hat{\alpha}_{i,j-1}$. Using sumcheck self-reducibility, this can be done by a logspace-uniform circuit of size t and depth $O(\log(T)) < O(\log^2(T))$.

Worst-case to rare-case reduction. By Proposition 4.7, each of the P_i's is the truth-table of a function that is worst-case to ρ_0-rare-case self-reducible, where $\rho_0 = h^{-c} \cdot \mathrm{polylog}(T)$, with error 2^{-h} and using logspace-uniform circuits of size t and depth $\mathrm{polylog}(T)$. This value of ρ_0 suffices by choosing a sufficiently small $\alpha \le 1/c'$, which guarantees that $t^{\alpha} \le h \le h^{c}$. ■

4.4 From bootstrapping systems to a targeted HSG

In this section we prove Proposition 4.4. We will directly prove the "moreover" part, under the stronger hypotheses (referring to logspace-uniformity and depth bounds); the proof of the basic claim (without the "moreover" part) is essentially identical, just ignoring depth bounds and logspace-uniformity. For convenience, we will denote the dimensions of the bootstrapping system in our hypothesis by $d \times T$ instead of $d' \times T'$.

For the proof we will need the following standard tools: The Nisan-Wigderson PRG [NW94] and the Goldreich-Levin [GL89] list-decoding algorithm for the Hadamard code. In the result statements below we assert that the reconstruction algorithm for the Nisan-Wigderson PRG is a uniform learning algorithm, following the classical observation of [IW98], and moreover assert that all the associated algorithms can be implemented by logspace-uniform circuits of bounded depth. The only non-standard thing in the latter efficiency requirement is that the circuits are logspace-uniform; we meet this requirement by constructing combinatorial designs in logspace (following [KM98; HR03] and an idea attributed to Salil Vadhan), so that the logspace algorithm that constructs the uniform circuit can "hardwire" the design into the circuit. Proofs of the two result statements appear in Appendices A.1 and A.4, respectively.

Theorem 4.8 (the NW PRG with reconstruction as a learning algorithm)**.** *There exists a universal constant $c > 1$, an oracle machine G, and a probabilistic oracle machine R_0, such that the following holds:*

1. **Generator:** *When given input $(1^{\ell}, 1^{M}, \gamma)$ such that $M \le 2^{(\gamma/c)\cdot\ell}$ oracle access to $h\colon \{0,1\}^{\ell} \to \{0,1\}$, the machine G runs in time $2^{c\cdot\ell/\gamma}$ and outputs a set of strings in $\{0,1\}^{M}$. Moreover, if γ is constant and ℓ, M are sufficiently large, then G can be implemented by logspace-uniform oracle circuits of size $2^{c\cdot\ell/\gamma}$ and depth $O(\log(M,\ell))$.*

2. **Reconstruction:** *When given input $(1^{\ell}, 1^{M}, \gamma)$ and oracle access to a $(1/M)$-distinguisher D for $G^{h}(1^{\ell}, 1^{M}, \gamma)$ and to h, the machine R_0 runs in time $M^{c} \cdot 2^{\gamma\cdot\ell}$, makes non-adaptive queries, and outputs with probability at least $1 - 2^{-3M}$ an oracle circuit that computes h on $1/2 + M^{-3}$ of the inputs when given access to D. The circuit that R_0 outputs has depth* $\mathrm{polylog}(M,\ell)$ *and makes just one oracle query. Moreover, if γ is a constant and ℓ, M are sufficiently large, then R_0 can be implemented by a logspace-uniform probabilistic oracle circuit of size $M^{c} \cdot 2^{\gamma\cdot\ell}$ and depth* $\mathrm{polylog}(M,\ell)$ *that makes non-adaptive queries.*

Theorem 4.9 (list-decoding the Hadamard code [GL89]). *For any time-computable $a\colon \mathbb{N} \to \mathbb{N}$ satisfying $a(\ell_0) \leq \ell_0$ and $\epsilon\colon \mathbb{N} \to (0, 1/2)$ there exists a transformation* $\mathtt{Had}$ *that maps any function $g\colon \{0,1\}^{\ell_0} \to \{0,1\}^{a(\ell_0)}$ to a Boolean function $\mathtt{Had}(g)\colon \{0,1\}^{\ell_0+a(\ell_0)} \to \{0,1\}$ such that the following holds.*

1. **Encoding:** *For every $x \in \{0,1\}^{\ell_0}$ and $z \in \{0,1\}^{a(\ell_0)}$ it holds that $\mathtt{Had}(g)(x,z) = \langle g(x), z\rangle = \bigoplus_{i\in[a(\ell_0)]} g(x)_i \cdot z_i$.*
2. **Decoding:** *There exists a logspace-uniform circuit* $\mathtt{GL}$ *of size* $\mathrm{poly}(\ell_0/\epsilon)$ *and depth* $\mathrm{polylog}(\ell_0/\epsilon)$ *that gets input 1^{ℓ_0} and outputs a probabilistic oracle circuit C of depth* $\mathrm{polylog}(\ell_0/\epsilon)$ *that satisfies the following. For every oracle $\widetilde{\mathtt{Had}}(g)$ that agrees with $\mathtt{Had}(g)$ on $1/2 + \epsilon$ of the inputs, the probability over the random coins of C and a uniform choice of $x \in \{0,1\}^{\ell_0}$ that $C^{\widetilde{\mathtt{Had}}(g)}(x) = g(x)$ is at least* $\mathrm{poly}(\epsilon)$.

We now describe the generator H_f and then later the reconstruction algorithm R. Throughout the description, the algorithms that we describe will be logspace-uniform circuits of bounded depth. We note that in intermediate stages of their execution, these algorithms will compute descriptions of certain circuits and then simulate these circuits; we will always bound the depth of the circuits whose descriptions are computed, in order to verify that the algorithm can then indeed simulate these circuits in small depth.[24]

The hitting-set generator H_f. Given $x \in \{0,1\}^n$, the algorithm H_f enumerates in parallel over $i \in [d]$ and for each i computes the string $P_i(x)$. Thinking of $P_i(x) \in [A]^T$ as a truth-table of a function $p_i\colon \{0,1\}^{\log(T)} \to [A]$, the algorithm computes the truth-table of the function $h_i = \mathtt{Had}(p_i)$, where $\mathtt{Had}$ is the encoding from Theorem 4.9. Note that h_i is a Boolean function over $\ell = \log(T) + \log(A) = (1 + o(1)) \cdot \log(T)$ bits, which means that its truth-table is of size $T^{1+o(1)}$. Finally, for the parameter $\gamma = \gamma(n) \in (0,1)$ chosen in our statement, the algorithm uses the generator G from Theorem 4.8 with parameters $(1^\ell, 1^M, \gamma)$ and with access to the function h_i, to output a set of strings of length M. (The hypothesis of Theorem 4.8 is satisfied by our assumption that $M \leq T^{\gamma/c''}$ for a sufficiently large constant $c'' \geq 1$.)

To bound the complexity of H_f, recall that $P_i(x)$ can be computed by a logspace-uniform circuit of depth $\widetilde{d}$ and size $\bar{T}$. Now, since each entry in the truth-table of h_i can be computed in time $\mathrm{polylog}(A)$ with non-adaptive access to the truth-table of p_i (i.e., to the string P_i), we can compute the entire truth-table of h_i by a logspace-uniform circuit of size $\mathrm{poly}(\bar{T})$ and depth $\mathrm{polylog}(A) \leq \mathrm{polylog}(T)$. Finally, the generator from Theorem 4.8 can be computed by a logspace-uniform circuit of size $T^{O(1/\gamma)}$ and depth $O(\log(M \cdot \ell)) = O(\log(T))$. Thus, for each $i \in [d]$ the corresponding output string can be printed by a logspace-uniform circuit of size $\mathrm{poly}(T^{1/\gamma}, \bar{T})$ and depth $\widetilde{d} + \mathrm{polylog}(T)$, and multiplying the size by d (since we enumerate over $i \in [d]$ in parallel) we still get a circuit of size at most $(T^{1/\gamma} + \bar{T})^{c'}$ for a sufficiently large universal constant c'.

[24]The potential issue here is that a logspace-uniform circuit C of bounded depth can potentially compute a description of a circuit C' of very large depth (in which case C would not be able to simulate C' in bounded depth). However, we will make sure that this does not happen in our specific constructions.

The reconstruction algorithm R. Let $D\colon \{0,1\}^M \to \{0,1\}$ be such that $\Pr_{r\in\{0,1\}^M}[D(r)=1] \geq 1/M$, but for every $i\in[d(n)]$ it holds that D rejects all the strings in the output set of $G^{h_i}(1^\ell,1^M,\gamma)$. The algorithm R gets input $x\in\{0,1\}^n$ and iteratively, for $i=0,...,d(n)$, it finds a small circuit that computes the function whose truth-table is $P_i(x)$. We will now describe the procedure, while accounting both for the complexity of implementing each iteration, and for the complexity of the circuit that each iteration produces.

First, the algorithm constructs a circuit C_0 that computes the function whose truth-table is P_0; by our assumption about P_0, this can be done by a logspace-uniform circuit of size t_0 and depth $\mathrm{polylog}(T)$, and also C_0 has size $\tilde{O}(t_0)$ and depth $\mathrm{polylog}(T)$. Then, for $i\in[d(n)]$, we start the i^{th} iteration with an oracle circuit C_{i-1} of size $|C_{i-1}|$ and depth $\mathsf{Depth}(C_{i-1})$ such that C_{i-1}^D computes p_{i-1}. The following lemma shows that, given C_{i-1} and oracle access to D, we can efficiently compute a small circuit C_i that computes p_i:

Lemma 4.10 (the i^{th} iteration: moving from a circuit for p_{i-1} to a circuit for p_i)**.** *There exists a universal constant $c_0>1$ such that the following holds. Given the circuit C_{i-1} and oracle access to D, we can compute with probability at least $1-1/T^2-3\cdot 2^{-M}$ an oracle circuit C_i that computes p_i when given oracle access to D. This can be done by a logspace-uniform probabilistic circuit of size $t^2\cdot T^{2\gamma}\cdot(M\cdot|C_{i-1}|)^{c_0}$ and depth $(\log(T)\cdot\mathsf{Depth}(C_{i-1}))^{c_0}$, and the circuit C_i is of size $\tilde{O}(t)\cdot M^{c_0}\cdot T^{2\gamma}$ and depth $\log(T)^{c_0}$.*

Proof. The algorithm will consist of three steps. Loosely speaking, these correspond to the Nisan-Wigderson reconstruction algorithm (as in Theorem 4.8); to the Goldreich-Levin list-decoding algorithm (as in Theorem 4.9), coupled with a process of weeding the output list to find a single "good" circuit; and to applying worst-case to rare-case self-reducibility. These three steps are depicted in the following three claims.

Claim 4.10.1 (the [NW94] reconstruction)**.** *Given the circuit C_{i-1} and oracle access to D, we can compute with probability at least $1-2^{-M}$ an oracle circuit $C_{i,1}$ such that $C_{\mathtt{NW}}=C_{i-1}^D$ computes h_i correctly on $1/2+M^{-3}$ of the inputs. This step can be implemented by a logspace-uniform probabilistic circuit of size $\mathrm{poly}(M,|C_{i-1}|)\cdot T^{2\gamma}\cdot t$ and depth $\mathrm{poly}(\log(T),\mathsf{Depth}(C_{i-1}))$,* [25] *and the circuit $C_{i,1}$ is of size $\mathrm{poly}(M)\cdot T^{2\gamma}$ and of depth $\mathrm{polylog}(M)$.*

Proof. We invoke the reconstruction algorithm from Theorem 4.8 for h_i with input $(1^\ell,1^M,\gamma)$. We answer its oracle queries to a distinguisher using D, and we answer its queries to h_i using the fact that $h_i(x,z)=\langle P_i(x),z\rangle$, the downward self-reducibility algorithm for P_i, and the circuit C_{i-1} such that C_{i-1}^D computes P_{i-1}. With probability at least $1-2^{-M}$ this yields an oracle circuit $C_{i,1}$ such that $C_{\mathtt{NW}}=C_{i,1}^D$ computes h_i correctly on $1/2+M^{-3}$ of the inputs.

To bound the complexity of this step, recall that the reconstruction algorithm can be implemented by a logspace-uniform probabilistic oracle circuit of size $\mathrm{poly}(M)\cdot 2^{\gamma\cdot\ell}\leq \mathrm{poly}(M)\cdot T^{2\gamma}$ and depth $\mathrm{polylog}(M)$, and the circuit C_{i-1} that it outputs has depth $\mathrm{polylog}(M\cdot\ell)=\mathrm{polylog}(M)$. The size of $C_{i,1}$ is trivially bounded by $\mathrm{poly}(M)\cdot T^{2\gamma}$. Now, using the downward self-reducibility of the bootstrapping system, each query to h_i can be answered by a

[25]In the bootstrapping systems that we construct in Proposition 4.3 the queries of the downward self-reducibility algorithm (as well as all worst-case to rare-case reduction) are actually *non-adaptive*. Thus, the term $\mathrm{polylog}(T)\cdot\mathsf{Depth}(C_{i-1})$ can actually be replaced by $\mathrm{polylog}(T)+\mathsf{Depth}(C_{i-1})$. We do not apply this optimization since it does not significantly improve the final parameters of the current construction.

logspace-uniform circuit of size t and depth $\text{polylog}(T)$ that makes queries to C_{i-1}^{D}. Also recall that the queries are non-adaptive. Hence, the circuit size of the reconstruction algorithm is $\left(\text{poly}(M) \cdot T^{2\gamma}\right) \cdot t \cdot \text{poly}(|C_{i-1}|)$ and its depth is $\text{polylog}(M)$. $\square$

Claim 4.10.2 (the [GL89] list-decoding, coupled with weeding the list)**.** *Given the circuit C_{i-1} and oracle access to $C_{\mathtt{NW}}$ and to D, we can compute with probability at least $1 - 2^{-M}$ an oracle circuit $C_{i,2}$ such that $C_{\mathtt{GL}} = C_{i,2}^{C_{\mathtt{NW}}}$ computes p_i on at least $\mu_{GL} \overset{\text{def}}{=\!=} \text{poly}(1/M)$ of the inputs. This step can be implemented by a logspace-uniform circuit of size $t \cdot \text{poly}(M, |C_{i-1}|)$ and depth $\text{poly}(\log(M), \mathtt{Depth}(C_{i-1}))$, and the circuit $C_{i,2}$ is of size $\text{poly}(M)$ depth $\text{polylog}(M)$.*

Proof. We invoke the decoding algorithm from Theorem 4.9 for $h_i = \mathtt{Had}(p_i)$ with parameter $\epsilon = M^{-3}$ and input $1^{\log(T)}$. This algorithm produces a probabilistic oracle circuit $C'_{i,2}$ such that the probability over input $y \in \{0,1\}^{\log(T)}$ and the internal randomness of $C'_{i,2}$ that $(C'_{i,2})^{C_{\mathtt{NW}}}(y) = p_i(y)$ is $\epsilon_{GL} = \text{poly}(1/M)$. This step can be implemented by a logspace-uniform circuit of size $\text{poly}(\log(T), M) = \text{poly}(M)$ and depth $\text{polylog}(M)$, and the circuit $C'_{i,2}$ is of size $\text{poly}(M)$ and depth $\text{polylog}(M)$.

Then, we perform the following experiment for $O(M/\epsilon_{GL}) = \text{poly}(M)$ trials in parallel:

1. Randomly choose a fixed random string for $C'_{i,2}$ and hard-wire it into the circuit, to obtain a deterministic circuit $C_{i,2}$.
2. Estimate the agreement of $C_{i,2}^{C_{\mathtt{NW}}}$ with p_i, up to error $\epsilon_{GL}/10$ and with confidence $1 - 2^{-M^2}$. To do so we sample $\text{poly}(M)$ inputs, and evaluate p_i and $C_{i,2}^{C_{\mathtt{NW}}}$ on each input. Computing p_i is done using the downward self-reducibility algorithm, the circuit C_{i-1}, and our oracle D.
3. Consider $C_{i,2}$ to be $\mathtt{good}$ if $C_{i,2}^{C_{\mathtt{NW}}}$ agrees with p_i on at least μ_{GL} of the inputs.

With probability at least $1 - 2^{-M}$, all the estimates are correct and at least one choice of random string yields a good deterministic circuit $C_{i,2}$. We proceed with the first good $C_{i,2}$ that we find among the trials (according to some predetermined efficient ordering of the trials), and denote $C_{\mathtt{GL}} = C_{i,2}^{C_{\mathtt{NW}}}$.

The latter procedure can be implemented by a logspace-uniform circuit of size $t \cdot \text{poly}(M, |C_{i-1}|)$ and depth $\text{poly}(\log(M), \mathtt{Depth}(C_{i-1}))$. This is since for each of the $\text{poly}(M)$ inputs (which are sampled in parallel) we apply a logspace-uniform circuit of size t with oracle access to C_{i-1} and to D. The circuit $C_{i,2}$ that it produces is of the same size and depth as $C'_{i,2}$ (i.e., size $\text{poly}(M)$ and depth $\text{polylog}(M)$), since we just hard-wired randomness into $C'_{i,2}$. $\square$

Claim 4.10.3 (worst-case to rare-case reducibility)**.** *Given the circuit C_{i-1} and oracle access to $C_{\mathtt{GL}}$ and to D, we can compute with probability at least $1 - 2^{-M} - 1/T^2$ a circuit $C_{i,3}$ such that $C_{i,3}^{C_{\mathtt{GL}}}$ computes p_i. This step can be implemented by a logspace-uniform probabilistic circuit of size $t^2 \cdot \text{poly}(M, |C_{i-1}|)$ and depth $\text{poly}(\log(T), \mathtt{Depth}(C_{i-1}))$ and $C_{i,3}$ is of size $t \cdot \text{poly}(M)$ and depth $\text{polylog}(T)$.*

Proof. Recall that p_i is sample-aided worst-case to ρ-rare-case reducible for $\rho = t^{-\alpha}$. Denoting $\mu_{GL} = M^{-k}$ for a universal constant k, note that

$$\rho = t^{-\alpha} \leq M^{-k} = \mu_{GL} ,$$

where we relied on the hypothesis that $M \leq t^{\alpha/c''}$ for a sufficiently large constant $c'' \geq k/\alpha$. We invoke the sample-aided reduction with input $1^{\log(T)}$ and a sample of $\text{poly}(M)$ labeled examples of p_i that we produce using the downward self-reducibility algorithm for p_i, the circuit C_{i-1}, and our access to D. With probability $1 - 2^{-t^{\alpha}} > 1 - 2^{-M}$ (where we relied again on the hypothesis $M \leq t^{\alpha/c''} \leq t^{\alpha}$) this step produces a probabilistic oracle circuit $C'_{i,3}$ of size t such that for every $y \in \{0,1\}^{\log(T)}$ we have that $\Pr\left[(C'_{i,3})^{C_{\texttt{GL}}}(y) = p_i(y)\right] \geq 2/3$.

This step can be implemented by a logspace-uniform circuit of size $t^2 \cdot \text{poly}(M, |C_{i-1}|)$ and depth $\text{poly}(\log(T), \mathsf{Depth}(C_{i-1}))$, and the circuit that it produces is of size $t \cdot \text{poly}(M)$ and depth $\text{polylog}(T)$. To see this, recall that by our assumption, the sample-aided worst-case to rare-case reduction can be implemented by a logspace-uniform circuit of size t and depth $\text{polylog}(T)$; and note that (as in the previous step) we can produce each of the t samples that it requires by a logspace-uniform circuit of size $t \cdot \text{poly}(|C_{i-1}|)$ and depth $\text{poly}(\log(M), \mathsf{Depth}(C_{i-1}))$.

Implementing naive error-reduction in $C'_{i,3}$, we decrease its error from $1/3$ to $1/\text{poly}(T)$ at the cost of increasing its size by a multiplicative factor of $O(\log(T))$ and its depth by an additive factor of $O(\log(T))$. We then randomly choose a fixed random string for $C'_{i,3}$ and hard-wire it; with probability at least $1 - 1/T^2$ we obtain a deterministic oracle circuit $C_{i,3}$ such that $C_{i,3}^{C_{\texttt{GL}}}$ computes p_i. This can be implemented by a logspace-uniform circuit of size $\tilde{O}(t) \cdot \text{poly}(M)$ and depth $\text{polylog}(T)$. □

Let us now see how the combination of the foregoing three steps yields the algorithm for the i^{th} iteration. The three steps of the i^{th} iteration above succeed with probability at least $1 - 1/T^2 - 3 \cdot 2^{-M}$. Assuming all the steps above succeeded, we have the following:

$C_{\texttt{NW}} = C_{i,1}^{D}$	computes h_i correctly on $1/2 + M^{-3}$ of the inputs
$C_{\texttt{GL}} = C_{i,2}^{C_{\texttt{NW}}}$	computes p_i correctly on $\mu_{GL} = \text{poly}(1/M)$ of the inputs
$C_{i,3}^{C_{\texttt{GL}}}$	computes p_i

This yields an oracle circuit C_i that computes p_i when given oracle access to D (by replacing the oracle gates in $C_{i,3}$ with $C_{i,2}$, replacing the oracle gates in $C_{i,2}$ with $C_{i,1}$, and keeping the oracle gates in $C_{i,1}$ intact). Note that the size of C_i is

$$|C_i| = \tilde{O}(t) \cdot \text{poly}(M) \cdot T^{2\gamma} ,$$

and its depth is $\text{polylog}(T)$. This is since $C_{i,3}$ is of size $\tilde{O}(t) \cdot \text{poly}(M)$, and $C_{i,2}$ is of size $\text{poly}(M)$, and $C_{i,1}$ is of size $\text{poly}(M) \cdot T^{2\gamma}$, and all three circuits are of depth $\text{polylog}(T)$.

Also, by accounting for the complexity of each of the three steps above, the entire i^{th} iteration can be implemented by a logspace-uniform circuit of size

$$t^2 \cdot \text{poly}(M, |C_{i-1}|) + \text{poly}(M, |C_{i-1}|) \cdot T^{2\gamma} \cdot t \leq t^2 \cdot T^{2\gamma} \cdot \text{poly}(M, |C_{i-1}|)$$

and depth $\text{poly}(\log(T), \mathsf{Depth}(C_{i-1}))$.

We stress that the constant powers hiding inside the poly notation in the size and depth bounds above (both for the circuit implementing the i^{th} iteration and for the circuit that it produces) are universal, arising from the universal constants in Theorem 4.8, Theorem 4.9, Proposition 3.10, and the cost of simulating a circuit of bounded depth (given its description) by a logspace-uniform circuit of bounded depth. Thus we will bound the polynomials in all these expressions by the polynomial power c_0 for a universal constant $c_0 > 1$. ■

After d applications of Lemma 4.10, with probability at least $1 - d \cdot \left(\frac{1}{T^2} + 3 \cdot 2^{-M}\right)$ this algorithm yields a circuit C_d such that C_d^D computes p_d. By evaluating C_d^D on the inputs corresponding to first n bits of $P_d(x)$, we obtain the value of $f(x)$.

To bound the overall complexity of the algorithm, we separate the first iteration $i = 1$ (in which $|C_0| \leq t_0$) from iterations $i = 2, ..., d$. The first iteration can be implemented by a logspace-uniform circuit of depth $\mathrm{polylog}(T)$ and of size

$$t^2 \cdot T^{2\gamma} \cdot M^{c_0} \cdot t_0^{c_0} ,$$

and the other $d - 1$ iterations can be implemented (together) by a logspace-uniform circuit of depth $(d-1) \cdot \mathrm{polylog}(T)$ and size

$$(d-1) \cdot t^2 \cdot T^{2\gamma} \cdot M^{c_0} \cdot (\tilde{O}(t) \cdot M^{c_0} \cdot T^{2\gamma})^{c_0} < (d-1) \cdot (t \cdot T^{\gamma} \cdot M)^{4c_0^2} .$$

Also accounting for the last step (of evaluating C_d on n inputs), the procedure in its entirety can be implemented by a logspace-uniform circuit of depth $d \cdot \mathrm{polylog}(T)$ and size

$$\begin{aligned} &t^2 \cdot T^{2\gamma} \cdot M^{c_0} \cdot t_0^{c_0} + (d-1) \cdot (t \cdot T^{\gamma} \cdot M)^{4c_0^2} + n \cdot (\tilde{O}(t) \cdot M^{c_0} \cdot T^{2\gamma})^{c_0} \\ &< (t \cdot T^{\gamma} \cdot M)^{4c_0^2} \cdot (d + n + t_0^{c_0}) , \end{aligned}$$

and this concludes the proof if we set the universal constant c' to be at least $4c_0^2$.

5 Non-black-box derandomization from "almost-all-inputs" hardness

In this section we use the reconstructive targeted HSG from Section 4 to prove various hardness-to-randomness results. In Section 5.1 we state hardness-to-randomness results for general probabilistic algorithms, and in particular prove Theorem 1.2. In Section 5.2 we state hardness-to-randomness results for restricted (probabilistic) circuit classes, and in particular prove Theorem 1.5. And in Section 5.3 we prove Theorem 1.4 for the "low-end" setting.

For some of our results (in Sections 5.1 and 5.3) we will need the following modified version of our reconstructive HSG from Proposition 4.5, which was mentioned prior to the proposition's statement. Compared to the latter HSG, the reconstruction algorithm in the following result can be more efficient, at the cost of a less efficient HSG; specifically, the overhead of T^{δ} in the reconstruction time can now be for a *subconstant* $\delta = o(1)$, at the cost of having an HSG with running time $T^{O(1/\delta)}$, and both the HSG and the reconstruction are not necessarily computable by logspace-uniform circuits of bounded depth.

Proposition 5.1 (a reconstructive targeted HSG, a version with low reconstruction overhead)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $f \colon \{0,1\}^n \to \{0,1\}^n$ be computable by logspace-uniform circuits of size $T(n)$ and depth $d(n)$, let $\delta \colon \mathbb{N} \to (0,1)$, and let $M \colon \mathbb{N} \to$*

$\mathbb{N}$ *such that*

$$c \cdot \log(T(n)) \le M(n) \le T(n)^{\delta(n)/c} ,$$
$$\delta(n) \ge c \cdot \frac{\log\log(T)}{\log(T)} .$$

Then, there exist a deterministic algorithm H_f and a probabilistic algorithm R that for every $x \in \{0,1\}^n$ satisfy the following:

1. **Generator.** *The generator H_f gets input x, runs in time $T^{O(1/\delta)}$, and outputs a set of M-bit strings.*

2. **Reconstruction.** *When R gets input x and oracle access to a function $D\colon \{0,1\}^M \to \{0,1\}$ such that $\Pr_{r\in\{0,1\}^M}[D(r) = 1] \ge 1/M$ but D rejects all the strings that $H_f(x)$ prints, it runs in time $(d + n^c) \cdot T^{\delta} \cdot M^c$ and outputs $f(x)$ with probability at least $1 - 1/M$.*

Proof. The proof is identical to the proof of Proposition 4.5, with only the following differences. First, we verify that the hypothesized lower bound $T^{\mu} \ge \log^{1/\alpha}(T)$ holds in Proposition 4.3 (the lower bound holds by our choice of $\mu = \delta/5c'$ and by our assumption about δ). Secondly, we do not claim that H_f and R are logspace-uniform circuits of bounded depth, so we just bound their running time (using the exact same bound as the size bound on the circuits). Thirdly, in our conclusion the running time of the generator H_f is $T^{O(1/\delta)}$, which is not necessarily polynomial in T. ■

Throughout this section, when we say that a probabilistic algorithm A computes a multi-output function f on input x, we mean that $\Pr[A(x) = f(x)] \ge 2/3$, where the probability is over the internal coin tosses of A.

5.1 Hardness-to-randomness tradeoffs and proof of Theorems 1.2 and 1.3

We now prove several hardness-to-randomness tradeoffs as a consequence of Propositions 4.5 and 5.1, and in particular we deduce Theorem 1.2. We first prove Claim 1.1, which asserts that "almost-all-inputs" lower bounds are necessary for derandomization. The proof is slightly more subtle than one might expect, since the hard function has multiple output bits but the derandomization hypothesis refers to machines with a single output bit.

Claim 5.2 (Claim 1.1, restated)**.** *If $pr\mathcal{BPP} = pr\mathcal{P}$, then for every $c \in \mathbb{N}$ there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in deterministic polynomial time such that f cannot be computed in probabilistic time n^c on* almost all inputs.

Proof. By our hypothesis, $pr\mathcal{BPTIME}[O(n^c)] \subseteq pr\mathcal{DTIME}[n^{c'}]$, for a sufficiently large constant c'. The function f gets input $x \in \{0,1\}^n$, and for each $i \in [n]$ it simulates the i^{th} Turing machine M_i for $n^{c'+2}$ steps on input x, and sets $f(x)_i$ to be the opposite of the i^{th} output bit of M_i; that is, $f(x)_i = 1 - M_i(x)_i$ (or $f(x)_i = 0$, in case M_i does not halt after $n^{c'+2}$ steps).

Assume towards contradiction that f can be computed on an infinite set $X \subseteq \{0,1\}^*$ of inputs by a probabilistic machine M_0 that runs in time n^c. Consider the probabilistic machine M_1 that takes as input a pair (x,i) of strings of identical length, and outputs $M_1(x,i) = M_0(x)_i$ (if $i > |x|$ or $M_0(x)$ does not output $|x|$ bits then $M_1(x,i) = 0$). Note that M_1 induces a promise-problem

$\Pi \in pr\mathcal{BPTIME}[O(n^c)]$, where the promise is that $x \in X$. By our derandomization hypothesis, $\Pi \in pr\mathcal{DTIME}[n^{c'}]$. Let M_2 be a deterministic machine that solves Π in time $n^{c'}$, and note that for every $x \in X$ and every $i \in [|x|]$ we have that $M_2(x)_i = f(x)_i$.

Consider the following procedure: Given input $x \in \{0,1\}^n$, we simulate M_2 on inputs $(x,1), ..., (x,n)$ and print the concatenation of its outputs. For every $x \in X$, this procedure prints $f(x)$ in time $O(n^{c'+1})$. Now, let i be the index of a Turing machine M_i that implements the foregoing procedure in time $O(n^{c'+1})$. By the definition of f, for every input $x \in \{0,1\}^*$ of sufficiently large length $n \geq i$ (in particular, for infinitely many inputs $x \in X$) we have that $f(x)_i = 1 - M_i(x)_i$. This is a contradiction. ■

We now state a very general tradeoff. Loosely speaking, and choosing nice parameters, the following result asserts that we can derandomize $pr\mathcal{RTIME}[M]$ using a function f that is hard for probabilistic time poly(M) on almost all inputs, with the following parameters: If f is computable by logspace-uniform circuits of size T and depth poly(M), then the derandomization time is polynomial in $2^{\log^2(T)/\log(M)}$. The foregoing parametrization is a special case, since the result is further parametrized by the depth d of circuits for f.

Theorem 5.3 (non-black-box derandomization, general version)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $T, M, d \colon \mathbb{N} \to \mathbb{N}$ such that $n \leq \max\{d(n), M(n)\} \leq T(n) \leq 2^{M(n)/c}$. Let $\Pi \in pr\mathcal{RTIME}[M]$, and let $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be an ensemble of distributions such that $\mathbf{x}_n$ is over $\{0,1\}^n$ and does not violate the promise of Π. Assume that for $\mu \colon \mathbb{N} \to [0,1)$ there exists a function $f \colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform circuits of size $T(n)$ and depth $d(n)$ such that for every probabilistic algorithm A running in time $d(n) \cdot M(n)^c$, the probability over $x \sim \mathbf{x}_n$ that $A(x)$ computes $f(x)$ is at most $\mu(n)$. Then, $\Pi \in \mathsf{heur}_{\mathbf{x},1-\mu}\text{-}pr\mathcal{DTIME}\left[2^{c\cdot(\log^2(T)/\log(M))}\right]$.*

In particular, if there exists $f \colon \{0,1\}^n \to \{0,1\}^n$ as above such that every probabilistic algorithm running in time $d(n) \cdot M(n)^c$ fails to compute f on almost all inputs, *then there exists a $(1/M)$-targeted HSG for time M with seed length $O(\log(T)^2/\log(M))$ and running time $2^{O(\log^2(T)/\log(M))}$, and consequently*

$$pr\mathcal{RTIME}[M(n)] \subseteq pr\mathcal{DTIME}\left[2^{c\cdot(\log^2(T)/\log(M))}\right] .$$

Proof. Let $\Pi \in pr\mathcal{RTIME}[M(n)]$, let M_Π be a probabilistic linear-time machine that solves Π, and let $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be an ensemble of distributions that do not violate the promise of Π. We instantiate Proposition 5.1 with the function f and with parameters T, d, M and with δ such that $M = T^{\delta/c_0}$, where c_0 is the universal constant from Proposition 5.1 (i.e., $\delta = c_0 \cdot \log(T)/\log(M)$). Note that the hypothesis of Proposition 5.1 is satisfied by our constraints on M and T and by our choice of δ.

Given input $x \in \{0,1\}^n$, we compute the set $H(x) = \left\{r_i \in \{0,1\}^{O(n)}\right\}_{i\in\{0,1\}^{O(\log(T)/\delta)}}$, and accept if and only if there exists $i \in \{0,1\}^{O(\log(T)/\delta)}$ such that M_Π accepts x with randomness r_i. The running time of this deterministic algorithm is $T^{O(1/\delta)} \cdot M(n) = 2^{O(\log^2(T)/\log(M))}$.

Note that foregoing algorithm never accepts "no" instance of Π, and thus may only err by rejecting "yes" instances. Assume towards a contradiction that for some $n \in \mathbb{N}$, with probability more than $\mu(n)$ over $x \sim \mathbf{x}_n$ it holds that x is a "yes" instance of Π that the foregoing algorithm rejects. Then, we can compute f on n-bit inputs with success probability more than $\mu(n)$ over

$x \sim \mathbf{x}_n$, as follows. Given x, we use the algorithm R from Proposition 5.1 with the function $D_x(r) = M_\Pi(x,r)$ as the distinguisher. By our assumption, for every $x \in S$ the function D_x accepts at least $1/2$ of its inputs, but rejects all the strings that $H(x)$ outputs. Hence, for every "yes" instance that the deterministic algorithm above rejects, our algorithm outputs $f(x)$ with probability $1 - 1/O(n) \gg 2/3$, in time at most

$$O(n + d(n)) \cdot T(n)^\delta \cdot M(n)^{c_0} = O(n + d(n)) \cdot M(n)^{2c_0} < d(n) \cdot M(n)^{3c_0} ,$$

which contradicts the hardness of f if we choose $c \geq 3c_0$.

The "in particular" part of the statement follows since the assumption that f is hard on almost all inputs implies that f is hard for all possible distributions $\mathbf{x}_n$ and with success bound $\mu(n) = 0$. In this case, for every time-M algorithm A and every fixed x of sufficiently large length such that $\Pr_r[A(x,r) = 1] \geq 1/2$, the proof above implies that there exists $r_i \in H(x)$ such that $A(x, r_i) = 1$. Indeed, it follows that the algorithm that gets input x and outputs $H(x)$ is a $(1/M)$-targeted HSG for time M. ■

We now prove Theorem 1.3, which assert a hardness-to-randomness tradeoff for derandomization of polynomial-time algorithms. The theorem follows by instantiating Theorem 5.3 with $M = \mathrm{poly}(n)$, and deducing derandomization of algorithms with two-sided error via the standard reduction by [Sip83; Lau83] of derandomization of $pr\mathcal{BPP}$ to *any* derandomization (i.e., not necessarily black-box) of $pr\mathcal{RP}$ (see, e.g., [BF99] and [GVW11] for an explanation).

Corollary 5.4 (non-black-box derandomization of $pr\mathcal{BPP}$; Theorem 1.3, restated)**.** *There exists a universal constant $c > 1$ such that the following holds. Assume that there exists a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform circuits of size $T(n)$ and depth $d(n)$ that cannot be computed in probabilistic time $d(n) \cdot n^c$ on* almost all inputs*. Then, we have that*

$$pr\mathcal{RP} \subseteq \bigcup_{a \geq 1} pr\mathcal{DTIME}[\bar{T}(n^a)]$$
$$pr\mathcal{BPP} \subseteq \bigcup_{a \geq 1} pr\mathcal{DTIME}[\bar{T}(\bar{T}(n^{c \cdot a})^a)] ,$$

where $\bar{T}(m) = 2^{c \cdot \log^2(T(m))/\log(m)}$.

Proof. Let $c > 1$ be the universal constant from Theorem 5.3. We can assume wlog that $T(n) \leq 2^{n/c}$, otherwise the conclusion is trivial. This allows us to instantiate Theorem 5.3 with $M(n) = O(n)$, and deduce that $pr\mathcal{RTIME}[O(n)] \subseteq pr\mathcal{DTIME}[\bar{T}(n)]$, where $\bar{T}(n) = 2^{c \cdot (\log^2(T)/\log(n))}$. The derandomization of $pr\mathcal{RP}$ follows by a padding argument (reducing any problem in $pr\mathcal{RTIME}[n^a]$ to $pr\mathcal{RTIME}[O(n)]$ by padding the input to length n^a).

For the derandomization of $pr\mathcal{BPP}$, we rely on the fact that for some fixed $k \in \mathbb{N}$, every problem in $pr\mathcal{BPTIME}[O(n)]$ is probabilistically reducible in probabilistic time $O(n^k)$ and with one-sided error to a corresponding problem in $pr\mathcal{RTIME}[O(n)]$ (see [Sip83; Lau83; BF99; GVW11]). Thus, for some constant $b \geq 1$ we have that $pr\mathcal{BPTIME}[O(n)] \subseteq pr\mathcal{DTIME}[\bar{T}(\bar{T}(n^b)^b)]$, which implies the derandomization of $pr\mathcal{BPP}$ (by a padding argument as above). ■

Remark 5.5. We suspect that the derandomization time of $2^{\log^2(T)/\log(n)}$ in Corollary 5.4 can be improved to be only $\text{poly}(T)$. The current overhead comes from applying the Nisan-Wigderson PRG to each row in the bootstrapping matrix, and in particular from the combinatorial designs underlying this PRG. More sophisticated constructions of HSGs and PRGs that avoid this particular overhead are well-known (e.g., by Shaltiel and Umans [SU05] and by Umans [Uma03]), and we suspect that using such constructions instead of the NW PRG might improve the overhead.

Remark 5.6. The hypothesis in Corollary 5.4 is that $f\colon \{0,1\}^n \to \{0,1\}^n$ is hard for *all* probabilistic algorithm that run in the prescribed time. However, to deduce the derandomization conclusion it suffices to assume that f is hard for *one particular algorithm*, which is the algorithm that is obtained when invoking the reconstruction procedure for f from Proposition 5.1 with the distinguisher being the standard machine solving the CAPP problem.[26] (This is the case because if the targeted HSG "fools" this machine, then we can derandomize all of $pr\mathcal{RP}$, relying on the completeness of the one-sided error version of CAPP for $pr\mathcal{RP}$.) Thus, to deduce the derandomization conclusion it suffices to analyze the hardness of f with respect to a single algorithm.

Our main result in this section (i.e., Theorem 1.2) is the special case of Corollary 5.4 with $T(n) = \text{poly}(n)$. As mentioned in Section 1, we state a more general version of the result, which refers to a hard function of any fixed polynomial depth $d(n) = \text{poly}(n)$ (rather than n^2).

Corollary 5.7 (polynomial-time non-black-box derandomization of $pr\mathcal{BPP}$; Theorem 1.2, restated)**.** *There exists $c > 1$ such that for every $k \in \mathbb{N}$ the following holds. Assume that there exists a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform circuits of polynomial size and depth n^k that cannot be computed in probabilistic time n^{k+c} on* almost all inputs. *Then $pr\mathcal{BPP} = pr\mathcal{P}$.*

Note that Corollary 5.7 follows immediately from Corollary 5.4 using the parameters $T(n) = \text{poly}(n)$ and $d(n) = n^k$ (in which case $\bar{T}(n) = \text{poly}(n)$).

A different type of hardness-to-randomness tradeoff. In the hardness-to-randomness results above, to derandomize $pr\mathcal{BPTIME}[M]$ we assumed a function that is hard for probabilistic time $\text{poly}(M)$ and that can be computed by uniform circuits of relatively small depth (i.e., circuits of depth d and size $T \gg d$). We now show a different type of tradeoff, where we considerably relax the assumption that the circuits have bounded depth, but we assume hardness for larger probabilistic time (closer to the circuit size T than to the time bound M that we want to derandomize).

This type of hardness-to-randomness tradeoff is particularly appealing in the setting of derandomization in superpolynomial time, since in this setting the assumption about the depth of the circuits can be almost completely eliminated. For simplicity, let us state just one nice instantiation of it, which refers to derandomization in *quasipolynomial* time. (Instantiations with other time bounds can be obtained using the same idea.)

Corollary 5.8 (superpolynomial-time non-black-box derandomization of $pr\mathcal{BPP}$)**.** *There exists a constant $c > 1$ such that for every constant $\epsilon > 0$ the following holds. Assume that there exists a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform circuits of quasipolynomial*

[26]That is, the distinguisher $D = D_x$ interprets the input x as a description of a circuit C_x, gets a random (or pseudorandom) string r, and outputs $C_x(r)$.

size $T(n) = 2^{\log^c(n)}$ *(where* $c > 1$*) and depth* $T(n)^{1-\epsilon}$ *that cannot be computed in probabilistic time* $T(n)^{1-\epsilon/4}$ *on* almost all inputs. *Then,*

$$pr\mathcal{BPP} \subseteq pr\mathcal{QP} \, ,$$

where $pr\mathcal{QP} = \cup_c pr\mathcal{DTIME}[2^{\log^c(n)}]$.

Proof. The proof is very similar in structure to that of Theorem 5.3 and of Corollary 5.4, so we focus on pointing out the differences in parameters. Given $\Pi \in pr\mathcal{RTIME}[O(n)]$, we instantiate Proposition 5.1 with parameters T, d, $M = O(n)$, and $\delta = \epsilon/2$. The derandomization of Π runs in time $\mathrm{poly}(T)$, which is quasipolynomial in n, and it succeeds since the hardness is for time larger than $(d + n^{c_0}) \cdot T^{\epsilon/2} \cdot n^c < T^{1-\epsilon+\epsilon/2} \cdot \mathrm{poly}(n) < T^{1-\epsilon/4}$. It follows (by padding) that $pr\mathcal{RP} \subseteq pr\mathcal{QP}$, and using [Sip83; Lau83] we deduce that $pr\mathcal{BPP} \subseteq pr\mathcal{QP}$. ■

5.2 Tight hardness-to-randomness results for low-depth circuits

In this section we extend our hardness-to-randomness results to the setting of derandomizing probabilistic low-depth circuits, and in particular logspace-uniform $\mathcal{NC}$ circuits. These results follow using the reconstructive HSG in Proposition 4.5, relying on the fact that both the HSG and the reconstruction can be computed by logspace-uniform circuits of low depth.

Let us first define logspace-uniform $\mathcal{NC}$ circuits and logspace-uniform probabilistic $\mathcal{NC}$ circuits. (Recall that the general notion of probabilistic circuits that we use was defined in Definition 3.5.)

Definition 5.9 (logspace-uniform $\mathcal{NC}$)**.** *For two constants* $i, c \in \mathbb{N}$*, we denote by* logspace-uniform-$pr\mathcal{NC}^i[n^c]$ *the class of promise problems solvable by families of logspace-uniform circuits of depth* $\log^i(n)$ *and size* n^c*.*

Definition 5.10 (logspace-uniform probabilistic $\mathcal{NC}$)**.** *We denote by* logspace-uniform-$pr\mathcal{BP} \cdot \mathcal{NC}^i[n^c]$ *the class of promise problems solvable by families of logspace-uniform* probabilistic *circuits of such depth and size that err with probability at most* $1/3$*. We denote by* logspace-uniform-$pr\mathcal{R} \cdot \mathcal{NC}^i[n^c]$ *the class of promise problems solvable by families of logspace-uniform* probabilistic *circuits of such depth and size that err only on "yes" instances (i.e., have one-sided error) and with probability at most* $1/2$

Referring to Definitions 5.9 and 5.10, when we omit the size parameter we implicitly refer to some (unspecified) polynomial size, and when we omit the depth parameter we implicitly refer to some (unspecified) polylogarithmic depth.

Our first result asserts that, for a *fixed* i, logspace-uniform $\mathcal{NC}^i$ circuits with *one-sided error* can be derandomized by logspace-uniform deterministic circuits, conditioned on a hardness hypothesis that corresponds to i. (Later on we will extend this to all i and to derandomization with two-sided error.) In more detail:

Proposition 5.11 (non-black-box derandomization of logspace-uniform $\mathcal{NC}$)**.** *There exists a universal constant* $c > 1$ *such that for any constant* $i \in \mathbb{N}$ *the following holds. Assume that for some* $j \in \mathbb{N}$ *there exists a function* $f : \{0,1\}^n \to \{0,1\}^n$ *computable by deterministic logspace-uniform* $\mathcal{NC}^j$ *circuits that cannot be computed by logspace-uniform probabilistic circuits of size* n^c *and of depth* $\log^{i+j+c}(n)$ *on* almost all inputs. *Then,*

$$\textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}^i \subseteq \textsf{logspace-uniform-}pr\mathcal{NC}\ .$$

Proof. We first show that $\textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}^{i+1}[O(n)] \subseteq \textsf{logspace-uniform-}pr\mathcal{NC}$, and then deduce the general statement by a padding argument. Fix $\Pi \in \textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}^{i+1}[O(n)]$, decidable by a circuit family $\{C_n\}$ that is printed by a uniform machine M_Π. Let f be the hard function from our hypothesis, and denote the size of the circuits for f by n^k and their depth by $\log^j(n)$. Let c' be the universal constant from Proposition 4.5.

Our deterministic algorithm uses the HSG H_f from Proposition 4.5 with the function f and with $\delta = c'/k$ and $M = O(n)$, then simulates C_n on input x and with each of the M-bit strings that H_f outputs (used as randomness for C_n), and finally outputs the majority value. By Proposition 4.5 and the properties of $\{C_n\}$, this algorithm is computable in logspace-uniform $\mathcal{NC}$.

Assuming towards a contradiction that this derandomization fails on infinitely many inputs x, for each such x we have that $C_{|x|}(x,\cdot)$ rejects all strings in the HSG's output set but accepts a random string with probability at least $1/2$. It follows that the reconstruction algorithm from Proposition 4.5 computes $f(x)$, with high probability. This reconstruction algorithm is computable by a logspace-uniform probabilistic circuit of size

$$\left(\text{polylog}(n) + n^{c'}\right)\cdot(n^k)^\delta\cdot O(n)^{c'} = O(n^{3c'})$$

and depth $\log^j(n)\cdot\log^{i+1}(n)\cdot\log^{c'}(n^k) < \log^{i+j+3c'}(n)$. This contradicts the hypothesized hardness of f if we set $c > 3c'$.

Finally, for any constant k let $\Pi \in \textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}^i[n^k]$. We define a padded version Π^{pad} whose "yes" instances are $\left\{(x, 1^{|x|^k-|x|}) : x \text{ is a yes instance of } \Pi\right\}$ and whose "no" instance are $\left\{(x, 1^{|x|^k-|x|}) : x \text{ is a no instance of } \Pi\right\}$. Observe that $\Pi^{\mathsf{pad}} \in \textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}^i[O(n)] \subseteq \textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}^{i+1}[O(n)]$, since we can compute in logspace a circuit that checks if the input is of the form $(x, 1^{|x|^k} - |x|)$ and that simulates the circuit for Π on the first part in the input (i.e., on x). Thus, by our hypothesis $\Pi^{\mathsf{pad}} \in \textsf{logspace-uniform-}pr\mathcal{NC}$. Then, given an input x, a logspace machine can print a circuit that pads its input x with $1^{|x|^k-|x|}$ and then simulates the circuit for Π^{pad}. Thus, $\Pi \in \textsf{logspace-uniform-}pr\mathcal{NC}$. ■

Let us now extend Proposition 5.11 to hold for all i and for derandomization with two-sided error. In the proof below, to leverage derandomization of $\textsf{logspace-uniform-}pr\mathcal{R}\cdot\mathcal{NC}$ to derandomization of $\textsf{logspace-uniform-}pr\mathcal{BP}\cdot\mathcal{NC}$ we rely on the fact that the classical reduction of [Sip83; Lau83] can be implemented by logspace-uniform $\mathcal{NC}$ circuits.

Corollary 5.12 (non-black-box derandomization of logspace-uniform $\mathcal{NC}$)**.** *There exists a universal constant $c > 1$ such that the following holds. Assume that for every $i \in \mathbb{N}$ there exists $j \in \mathbb{N}$ and a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by deterministic logspace-uniform $\mathcal{NC}^j$ circuits that cannot be computed by logspace-uniform probabilistic circuits of size n^c and of depth* $\log^{i+j+c}(n)$ *on* almost all inputs. *Then,*

$$\textsf{logspace-uniform-}pr\mathcal{BP}\cdot\mathcal{NC} \subseteq \textsf{logspace-uniform-}pr\mathcal{NC}\ .$$

Proof. Invoking Proposition 5.11 with all $i \in \mathbb{N}$, we deduce that logspace-uniform-$pr\mathcal{R} \cdot \mathcal{NC} \subseteq$ logspace-uniform-$pr\mathcal{NC}$. Now, let $\Pi \in$ logspace-uniform-$pr\mathcal{BP} \cdot \mathcal{NC}^i[n^k]$ for some constant k, solvable by a logspace-uniform circuit family $\{C_n\}$. We implement the standard reduction of [Sip83; Lau83] in logspace-uniform $\mathcal{NC}$, as follows. Consider the logspace-uniform circuit family $\{C'_n\}$ such that C'_n simulates C_n while implementing naive error-reduction, to reduce the error below 2^{-n}; note that $\{C'_n\}$ is also logspace-uniform, and of size $n^{k'}$ for some $k' > k$ and depth $O(\log^i(n))$.

Now, consider the following promise problem Π'. The valid inputs are of the form $(x, s_1, ..., s_{|x|^{k'}})$ where each s_i is a string of length $|x|^{k'}$. The "yes" instances are such that for every $r \in \{0,1\}^{|x|^{k'}}$ there exists $i \in [|x|^{k'}]$ for which $C'_{|x|}(x, r \oplus s_i) = 1$, and the "no" instances are such that for at least half of the $r \in \{0,1\}^{|x|^{k'}}$ it holds that $C'_{|x|}(x, r \oplus s_i) = 0$ for all $i \in [|x|^{k'}]$. Relying on the properties of $\{C'_n\}$ and on the fact that we can check the s_i's in parallel, the problem Π' is solvable by a logspace-uniform family of probabilistic circuits of size $n^{O(k')}$ and depth $O(\log^i(n))$ with one-sided error. Hence, Π' is also solvable by a logspace-uniform family $\{C''_n\}$ of *deterministic* circuits of size $n^{k''}$ and depth $\log^{i'}(n)$, for some constants $k'', i' \in \mathbb{N}$.

It follows that $\Pi \in$ logspace-uniform-$pr\mathcal{R} \cdot \mathcal{NC}$. To see this, consider a circuit family that gets input x, chooses $(s_1, ..., s^{k'}_{|x|})$ at random, and simulates the circuit C''_n (where $n = |x| + |x|^{2k'}$) on input $(x, s_1, ..., s_{|x|^{k'}})$. The well-known analysis (see, e.g., [Gol08, Theorem 6.9]) shows that this circuit family solves Π with one-sided error of at most $1/2$. Also, this circuit family is of polynomial size and of polylogarithmic depth. Hence, $\Pi \in$ logspace-uniform-$pr\mathcal{NC}$. ■

We complement Proposition 5.11 by showing that, analogously to Claim 5.2, functions that are hard on almost all inputs are necessary for derandomization of logspace-uniform $\mathcal{NC}$ circuits. This result relies on diagonalization, and in particular on diagonalizing against logspace-uniform circuits by other logspace-uniform circuits. To do so we need to quantify the "amount of logspace" used to construct the circuit, and so we introduce the following definitions.

Definition 5.13 (α-logspace-uniformity). *For a constant $\alpha \geq 1$, we say that a circuit family $\{C_n\}$ of size $T(n)$ is α-logspace-uniform if there exists a Turing machine M of space complexity $\alpha \cdot \log(T(n))$ such that $M(1^n)$ prints C_n.*

The following diagonalization-based proof is similar to that of Claim 5.2, but with the additional complication that the diagonalizing function must be computable by logspace-uniform circuits of bounded depth even when it simulates logspace machines whose output is not necessary a bounded-depth circuit.

Proposition 5.14 (almost-all-inputs hardness is necessary for derandomization of $\mathcal{NC}$). *Assume that logspace-uniform-$pr\mathcal{BP} \cdot \mathcal{NC} \subseteq$ logspace-uniform-$pr\mathcal{NC}$. Then, for every $\alpha \geq 1$ and $c_1, c_2 \in \mathbb{N}$ there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable by logspace-uniform circuits of polylogarithmic depth and polynomial size such that f cannot be computed by α-logspace-uniform probabilistic circuits of size n^{c_1} and depth $\log^{c_2}(n)$ on* almost all inputs.

Proof. The hard problem f is defined using two sufficiently large constants d_1 and d_2 that depend on c_1 and c_2 and will be determined in a moment. For every $x \in \{0,1\}^n$ we will print a circuit $\{0,1\}^n \to \{0,1\}^n$ that consists of n separate sub-circuits $C_1, ..., C_n$, one per each output bit $i \in [n]$. For each $i \in [n]$,

1. Let M_i be the i^{th} Turing machine (according to some efficient enumeration). We check that $M_i(1^n)$ uses $\alpha \cdot \log(n^{d_1})$ space and halts after $2^{\alpha \cdot \log(n^{d_1})}$ steps.

2. We print a circuit C_i that has the output of $M_i(1^n)$, denoted C'_i, hard-wired. The circuit C_i tries to simulate $C'_i(x)$ assuming that C'_i is a circuit of size n^{d_1} and depth $\log^{d_2}(n)$.

3. If the simulation of $C'_i(x)$ succeeds, then $C_i(x) = 1 - C'_i(x)$; otherwise, $C_i(x) = 0$.

Note that the algorithm for f can be computed by a logspace-uniform circuit family of polynomial size and of polylogarithmic depth. Assume towards a contradiction that f can be computed on an infinite set $X \subseteq \{0,1\}^*$ of inputs by an α-logspace-uniform probabilistic circuit family $\{F_n\}$ of size $T = T(n) = n^{c_1}$ and depth $\log^{c_2}(n)$, and let M_i be a Turing machine that prints this circuit family using $\alpha \cdot \log(T)$ space.

Let M'_i be a logspace machine that prints a circuit family $\{F'_n\}$ in which each F'_n gets input (x, i), simulates F_n on x, and prints the i^{th} bit of $F_n(x)$. Consider the promise-problem of mapping $(x, i) \mapsto f(x)_i$ where the promise is that $x \in X$. By our derandomization hypothesis, for some $d_1, d_2 \in \mathbb{N}$ there exists a logspace machine M''_i that prints a family $\{F''_n\}$ of *deterministic* circuits of size n^{d_1} and depth $\log^{d_2}(n)$ that agree with F'_n on every $x \in X$ and $i \in [|x|]$. If we use these values d_1 and d_2 in the definition of f above, we obtain a contradiction. ■

By combining Propositions 5.11 and 5.14 we obtain the following "near-equivalence" result, in which there is a very small gap between a hypothesis that suffices to derandomize logspace-uniform $\mathcal{NC}$ and a hypothesis that is necessary for doing so. The following statement implies Theorem 1.5.

Corollary 5.15 (non-black-box derandomization of $\mathcal{NC}$)**.** *There exists a universal constant $c > 1$ such that, considering the following statements, we have that (1) $\Rightarrow$ (2) $\Rightarrow$ (3).*

1. *(**Sufficient lower bound.**) For every sufficiently large $i \in \mathbb{N}$ there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in logspace-uniform deterministic $\mathcal{NC}^{(.99 \cdot i)}$ that cannot be computed by logspace-uniform probabilistic $\mathcal{NC}^i[n^c]$ on* almost all inputs*.*

2. *(**Derandomization.**)* logspace-uniform-$pr\mathcal{BP} \cdot \mathcal{NC} \subseteq$ logspace-uniform-$pr\mathcal{NC}$.

3. *(**Necessary lower bound.**) For every $i \in \mathbb{N}$ and $\alpha \geq 1$ there exists $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in logspace-uniform deterministic $\mathcal{NC}$ that cannot be computed in α-logspace-uniform probabilistic $\mathcal{NC}^i[n^c]$ on* almost all inputs*.*

Proof. The implication (2) $\Rightarrow$ (3) follows immediately from Proposition 5.14. To see that (1) $\Rightarrow$ (2), by Proposition 5.11 it suffices to show, for every fixed $i_0 \in \mathbb{N}$, a function f in logspace-uniform $\mathcal{NC}^j$ for some $j \in \mathbb{N}$ that cannot be computed by logspace-uniform probabilistic circuits of size n^c and depth $\log^{i_0+j+c}(n)$ on almost all inputs. Such a function exists by our hypothesis, taking j to be sufficiently large such that $j + (i_0 + c) < .99 \cdot j$. ■

5.3 "Low-end" derandomization from hard functions without structural constraints

We now prove Theorem 1.4, which deduces a fixed-exponential-time derandomization of $\mathcal{RP}$ from the existence of a hard function f without any structural constraints on f (i.e., we do not need to assume that f has logspace-uniform low-depth circuits).

Towards presenting the result, we say that a probabilistic algorithm A computes σ on input $x \in \{0,1\}^n$ with zero error and success probability β if $A(x)$ outputs either σ or $\perp$ and $A(x)$ outputs σ with probability at least β. We will also need the following definition, which relaxes "almost-all-inputs" hardness to only require failure on all inputs of certain lengths.

Definition 5.16 (infinitely-often almost-all-inputs hardness)**.** *We say that a function f is hard for a class $\mathcal{C}$ of probabilistic algorithms* infinitely often on almost all inputs *if for every $C \in \mathcal{C}$ there exists an infinite set $S \subseteq \mathbb{N}$ such that for every $n \in S$ and $x \in \{0,1\}^n$ it holds that* $\Pr[C(x) = f(x)] < 2/3$.

Analogously to the shorthand notation `i.o.` for "infinitely-often", we will use the shorthand notation `i.o.-aai` to refer to "infinitely-often almost-all-inputs" hardness. The following result, which implies Theorem 1.4, asserts that if there exists a function in $\mathcal{EXP}$ that is hard for $\mathcal{BPP}$ infinitely-often almost-all-inputs, then $\mathcal{RP}$ can be derandomized in fixed exponential time; and if $\mathcal{RP}$ can be derandomized in fixed exponential time, then there exists a function in $\mathcal{EXP}$ that is hard for $\mathcal{RP}$ infinitely-often almost-all-inputs. Indeed, the only gap between the hardness hypothesis and the hardness that we conclude from derandomization is that the former is for $\mathcal{BPP}$ whereas the latter is for $\mathcal{RP}$.

Theorem 5.17 (a near-equivalence in the "low-end" setting)**.** *Considering the following three statements, we have that (1) $\Rightarrow$ (2) $\Rightarrow$ (3).*

1. *(**$\mathcal{EXP}$ is `i.o.-aai` hard for $\mathcal{BPP}$.**) There exists a universal constant $c_1 \geq 1$ and a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in time $2^{n^{c_1}}$ such that f is hard for all polynomial-time probabilistic algorithms infinitely often on almost all inputs.*

2. *(**Infinitely-often non-trivial derandomization for** $pr\mathcal{RP}$**.**) There exists a universal constant $c_2 \geq 1$ such that $pr\mathcal{RP} \subseteq$ `i.o.`$\mathcal{DTIME}[2^{n^{c_2}}]$.*

3. *(**$\mathcal{EXP}$ is `i.o.-aai` hard for $\mathcal{ZPP}$.**) There exists a universal constant $c_3 \geq 1$ and a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in time $2^{n^{c_3}}$ such that f is hard for all polynomial-time probabilistic* zero-error *algorithms infinitely often on almost all inputs.*

Proof of (1) $\Rightarrow$ (2). First note that we can assume that $\mathcal{EXP} \subset \mathcal{P}/\text{poly}$. This since otherwise, by [NW94; BFN+93] we have that $pr\mathcal{BPP} \subseteq$ `i.o.`$pr\mathcal{SUBEXP}$ [NW94; BFN+93], which in particular implies that $pr\mathcal{RP} \subseteq$ `i.o.`$pr\mathcal{DTIME}[2^n]$.

Now, recall that $\mathcal{EXP} \subset \mathcal{P}/\text{poly}$ implies that $\mathcal{EXP} \subset \mathcal{MA}$ [BFL91; BFN+93]. Assuming that Item (1) holds for a constant $c_1 > 1$, we have that $\mathcal{DTIME}[2^{n^{c_1}}] \subseteq \mathcal{MA}$. This further implies that $\mathcal{DTIME}[2^{n^{c_1}}]$ is contained in $\mathcal{MATIME}[n^d]$ for a constant $d > 1$ (since $\mathcal{DTIME}[2^{n^{c_1}}]$ has a complete problem under linear-time reductions).

Note that any function in $\mathcal{MATIME}[n^d]$ can be computed by a log-space uniform circuit family of size $T_0(n) = 2^{O(n^d)}$ and depth $\log(T_0)$.[27]. By Item (1), there is a function $f\colon \{0,1\}^n \to \{0,1\}^n$ computable in time $2^{n^{c_1}}$ such that there is an infinite subset $S \subseteq \mathbb{N}$ for which f is instance-wise hard for all polynomial-time probabilistic algorithms infinitely often. Consider the function $f_{\mathsf{idx}}\colon \{0,1\}^n \times [n] \to \{0,1\}$ that maps (x,i) to $f(x)_i$. With a natural Boolean encoding of $[n]$, we have that f_{idx} is computable in $2^{n^{c_1}}$ time using the algorithm for f. Hence, by the discussion above, $f_{\mathsf{idx}} \in \mathcal{MATIME}[n^d]$ and hence it can be computed by a logspace-uniform circuit family of size at most $T_0(2n)$ and depth $\log T_0(2n)$. This implies that for some $T(n) = 2^{O(n^d)}$, f can be computed by logspace-uniform circuit family of size at most T and depth $\log T$.

Now, fix an arbitrary $pr\mathcal{RP}$ problem $\Pi = (\Pi_{\mathsf{yes}}, \Pi_{\mathsf{no}})$ solvable in time n^t, where $t \geq 1$ is a constant. That is, there is a deterministic Turing machine N taking two inputs x and y satisfying $|y| = n^t$, such that (1) $\Pr_{y \in \{0,1\}^{n^t}}[N(x,y) = 1] \geq 1/2$ for $x \in \Pi_{\mathsf{yes}}$ and (2) $\Pr_{y \in \{0,1\}^{n^t}}[N(x,y) = 1] = 0$ for $x \in \Pi_{\mathsf{no}}$. For convenience, we also define the set $A_x := \{y \in \{0,1\}^{n^t} : N(x,y) = 1\}$.

Let c be the constant in Proposition 5.1, let $M(n) = n^t$, and $\delta(n) = \tau \cdot (\log\log T(n)/\log T(n))$ for a large enough constant $\tau \geq 1$ so that $M(n) \leq T(n)^{\delta/c} = O(n^d)^{\tau/c}$ and $\log T(n) = O(n^d) \leq T(n)^{\delta/c}$. Note that without loss of generality, we can assume t is a large enough constant so that $M(n) = n^t > c \log T(n)$. Then, by Proposition 5.1 there is a generator H_f running in $T^{O(1/\delta)} = 2^{O(n^{d^2})}$ time such that, for every $x \in \Pi_{\mathsf{yes}}$ the following holds: If $H_f(x)$ does not hit the set A_x, plugging $N(x,\cdot)$ as the oracle of the reconstruction algorithm R, the resulting algorithm $R^{N(x,\cdot)}(x)$ computes $f(x)$ in $(\log T + n^c) \cdot T^\delta \cdot M^c \cdot n^t \leq n^\kappa$ time with probability at least $1 - 1/n$, for a constant $\kappa = \kappa(t,c,d)$.

Now consider the following algorithm A for solving Π: Given an input $x \in \{0,1\}^n$, enumerate all outputs y of the generator H_f; if any of them satisfy $N(x,y) = 1$ $A(x)$ output 1, and otherwise output 0. By Item (1), f is hard for the algorithm $R^{N(x,\cdot)}(x)$ infinitely often on almost all inputs. Let S be the corresponding infinite subset of $\mathbb{N}$ on which f is hard for $R^{N(x,\cdot)}(x)$. In the following, we show that for all $n \in S$, A computes Π on n-bit inputs.

Otherwise, for infinitely many input length $n \in S$, A fails to compute Π on some n-bit input. Note that A never errs on "no" instances, so this means that for every $n \in S$, there exists an input $x_n \in \Pi_{\mathsf{yes}} \cap \{0,1\}^n$ (for x_n in neither Π_{no} nor Π_{yes}, A is allowed to output anything) such that $H_f(x_n)$ does not hit the set A_{x_n}. By previous discussions, this means that $R^{N(x_n,\cdot)}(x_n)$ computes $f(x_n)$ with probability at least $1 - 1/n$, for infinitely many $n \in S$ and the corresponding $x_n \in \{0,1\}^n$, contradicting to the fact that f is hard for the algorithm $R^{N(x,\cdot)}(x)$ infinitely often on almost all inputs. ■

Proof of (2) ⇒ (3). Assume that Item (2) holds for a constant $c_2 \geq 1$. We define a function $f\colon \{0,1\}^n \to \{0,1\}^n$ as follows:

1. Given an input $x \in \{0,1\}^n$ and an output index $i \in [n]$, let M_i be the i-th deterministic Turing machine.

2. Simulate M_i on x for $2^{n^{c_2+1}}$ steps to obtain its output $\sigma \in \{0,1\}^n$. If M_i does not terminate in $2^{n^{c_2+1}}$ steps or its output does not have length exactly n, we set $\sigma = 0^n$.

[27]This is because we can construct a circuit that enumerates all possible n^d-length proofs in parallel, and verifies every proof deterministically by enumerating all possible n^d-bit strings as randomness in parallel. Note that the n^d-time verifier can be simulated by a logspace-uniform circuit of depth $O(n^d)$, using the standard tableau method.

3. Otherwise, set $f(x)_i = 1 - \sigma_i$.

Let $c_3 \geq 1$ be a constant such that f is computable in $2^{n^{c_3}}$ time (e.g., set $c_3 = c_2 + 2$), and let A be a polynomial-time probabilistic algorithm. We first establish the following claim.

Claim 5.18. *There is a $2^{n^{c_2+1}}$-time computable function $f_A \colon \{0,1\}^n \to \{0,1\}^n$ such that for infinitely many input lengths n and for every $x \in \{0,1\}^n$, if $A(x)$ computes $\sigma \in \{0,1\}^n$ with zero error and success probability at least $2/3$, then $f_A(x) = \sigma$.*

Proof. We first consider the following promise problem Π_A: On an input of length m, letting $n = \lfloor m/2 \rfloor$, it treats the first n bits as an input $x \in \{0,1\}^n$, the next $\lceil \log n \rceil$ bits as an index $i \in [n]$[28], and ignores the rest of the input. We say that an input (x, i) is in the promise of Π_A if it satisfies the following: There exists an output $\sigma \in \{0,1\}^n$ such that $\Pr[A(x) = \sigma] \geq 2/3$ and $\Pr[A(x) \in \{\sigma, \perp\}] = 1$. For such an (x, i) in the promise of Π_A, letting σ be the corresponding high-probability output of $A(x)$, we simply define $\Pi_A(x, i) = \sigma_i$. By a straightforward simulation algorithm, we have that $\Pi_A \in pr\mathcal{ZPP} \subseteq pr\mathcal{RP}$.

By our hypothesis in Item (2), there is an infinite subset $S \subseteq \mathbb{N}$ and a $2^{n^{c_2}}$-time algorithm M_A such that for every $m \in S$ it holds that M_A solves Π_A for all inputs of length m. Now we consider the following two cases:

1. There are infinitely many even integers in S. In this case, we define f_A as follows: Given input $x \in \{0,1\}^n$, for each $i \in [n]$, we set $f_A(x)_i = M_A(x, i, 0^{n - \lceil \log n \rceil})$.

2. There are finitely many even integers in S. Note that this implies there are infinitely many odd integers in S. In this case, we define f_A as follows: Given input $x \in \{0,1\}^n$, for each $i \in [n]$, we set $f_A(x)_i = M_A(x, i, 0^{n+1-\lceil \log n \rceil})$.

It is straightforward to verify that f_A in either cases satisfy the requirement. □

Now we are ready to show that f satisfies Item (3). Let A be a probabilistic randomized algorithm, and let f_A be the corresponding $2^{n^{c_2+1}}$-time algorithm guaranteed by Claim 5.18. Suppose that f_A is implemented by the i-th deterministic Turing machine. Then by Claim 5.18, for infinitely many input lengths $n \geq i$ and every $x \in \{0,1\}^n$, if $A(x)$ computes $\sigma \in \{0,1\}^n$ with zero error and success probability at least $2/3$, then $f(x)_i = 1 - \sigma_i \neq \sigma_i$. Therefore, f is hard for all polynomial-time probabilistic *zero-error* algorithms infinitely often on almost all inputs. ■

6 Non-black-box derandomization from "non-batch-computability"

In this section we show our uniform hardness-to-randomness tradeoff for the setting of superfast derandomization, and in particular prove Theorems 1.6 and 1.7. For convenience we define the following notion of a probabilistic algorithm that *approximately-prints* a multi-output function.

Definition 6.1 (approximately-printing a function). *Let A be a probabilistic algorithm, let $g \colon \{0,1\}^n \to \{0,1\}^k$, and let $x \in \{0,1\}^n$. For $\alpha \in [0,1]$, we say that A approximately-prints $g(x)$ with error α if with probability at least $1 - \alpha$ it holds that $\Pr_{i \in [k]}[A(x)_i = g(x)_i] \geq 1 - \alpha$.*

[28]If these bit correspond to an integer larger than n, Π_A truncates it to n.

In Section 6.1 we show a generic construction of a reconstructive targeted PRG, and prove that it can be instantiated with a suitable hard function to obtain superfast derandomization. In Section 6.2 we use the foregoing results to prove Theorems 1.6 and 1.7 (in particular, Section 6.2 includes the formal definitions of the direct-product hypothesis underlying Theorem 1.6). And in Section 6.3 we prove that a "non-batch-computable" function is *necessary* for superfast derandomization in the natural special case of highly-uniform formulas.

6.1 Derandomization from non-batch-computable functions

The following construction of a reconstructive targeted PRG underlies our main results. Loosely speaking, the targeted PRG G is based on a function $g\colon \{0,1\}^n \to \{0,1\}^k$ such that the individual bits of g (i.e., the mapping $(x,i) \mapsto g(x)_i$) can be computed in time T', and the result below shows a reconstruction algorithm that, given access to x and to a distinguisher D_x for $G(x)$, approximately-prints $g(x)$ with error α' in time $T' \cdot n^{\beta'}$, where α' and β' are arbitrarily small constants.

Proposition 6.2 (a reconstructive targeted PRG). *For every $\alpha', \beta' > 0$ and sufficiently small $\eta = \eta_{\alpha',\beta'} > 0$ the following holds. Let $T, k\colon \mathbb{N} \to \mathbb{N}$ be time-computable functions such that $T(n) \geq n$, and let $g\colon \{0,1\}^n \to \{0,1\}^k$ (where we denote $k = k(n)$) such that the mapping of $(x,i) \in \{0,1\}^n \times \{0,1\}^k$ to $g(x)_i$ is computable in time $T'(n)$. Then, there exist a deterministic algorithm G_g and a probabilistic algorithm R that for every $x \in \{0,1\}^n$ satisfy the following:*

1. **Generator.** *When G_g gets as input $x \in \{0,1\}^n$ and $\eta > 0$, it runs in time $k \cdot T'(n) + \text{poly}(k)$ and outputs a set of strings in $\{0,1\}^{k^\eta}$.*

2. **Reconstruction.** *When R gets as input $x \in \{0,1\}^n$ and $\eta > 0$, and gets oracle access to a function $D_x\colon \{0,1\}^{k^\eta} \to \{0,1\}$ that $(1/k^\eta)$-distinguishes the uniform distribution over the output-set of $G_g(x,\eta)$ from a truly uniform string, it runs in time $\tilde{O}(k^{1+\beta'}) + k^{\beta'} \cdot T'(n)$, makes $\tilde{O}(k^{1+\beta'})$ queries to D_x, and with probability at least $1 - 2^{-k^\eta}$ outputs a string that agrees with $g(x)$ on at least $1 - \alpha'$ of the bits.*

Proof. In our proof we will use the following reconstructive PRG, which is based on a combination of parts from the classical constructions of Nisan and Wigderson [NW94] and of Impagliazzo and Wigderson [IW99]. The PRG gets as input a k-bit truth-table of a function $\{0,1\}^{\log(k)} \to \{0,1\}$, outputs a set of $\text{poly}(k)$ strings of length k^η for a sufficiently small $\eta > 0$, and the crucial property for us is that its reconstruction algorithm is both *uniform* and of *very small time complexity* $k^{\beta'}$.

Theorem 6.3 (the PRG of [NW94; IW99] with reconstruction as a high-accuracy learning algorithm). *For every two constants $\alpha', \beta' \in (0,1)$ there exist an oracle machine $G^{\mathtt{IW}}$ and a probabilistic oracle machine $R^{\mathtt{IW}}$ such that for every function $g\colon \{0,1\}^{\log(k)} \to \{0,1\}$ and sufficiently small $\eta = \eta_{\alpha',\beta'} > 0$ the following holds.*

- **Generator:** *When given input $(1^k, \eta)$ and oracle access to g, the machine $G^{\mathtt{IW}}$ runs in time* $\text{poly}(k)$ *and outputs a set of strings in $\{0,1\}^m$, where $m = k^\eta$.*

- **Reconstruction:** *When given input $(1^k,\eta)$ and oracle access to a $(1/m)$-distinguisher D for $G^g(1^k,\eta)$ and to g, the machine $R^{\mathtt{IW}}$ runs in time $O(k^{\beta'})$ and with probability at least $1-2^{-2m}$ outputs outputs an oracle circuit that agrees with g on $1-\alpha'$ of the inputs when given access to D.*

The proof of Theorem 6.3 is based on well-known constructions and observations from [NW94; IW98; IW99], but it involves many low-level details, so we defer the full proof to Appendix A. In high-level, the PRG first encodes the function by the efficient derandomized direct-product construction of [IW99] and by the Hadamard encoding, and then applies the PRG construction of [NW94] instantiated with very small output length k^η. To design a very fast uniform reconstruction algorithm, we rely on the observation of [IW98] that the reconstruction algorithm for the PRG of [NW94] is a uniform learning algorithm, and further note that when the output length is small (i.e., k^η as in our setting), then the latter algorithm is also very fast (see Theorem A.4 for details). Similarly, the list-decoding algorithms for the the direct-product construction of [IW99] and for the Hadamard encoding are both very fast uniform algorithms (see Theorem A.5 for details). All these algorithms succeed with probability $1/\mathrm{poly}(m)$, and we repeat them to produce a list of candidate circuits that with high probability contains a circuit computing the initial function. We can then use our oracle access to the initial function in order to "weed" these lists and find a circuit that agrees with the function on $1-\alpha'$ of the inputs. See Appendix A for details.

Both the generator $G^{\mathtt{IW}}$ and the reconstruction $R^{\mathtt{IW}}$ in Theorem 6.3 are auxiliary protocols for the targeted generator G and reconstruction algorithm R that we construct. Specifically, G and R will instantiate $G^{\mathtt{IW}}$ and $R^{\mathtt{IW}}$ (respectively) with the k-bit truth-table $g(x)\in\{0,1\}^k$ that is a function of an n-bit input x, where x is given as explicit input both to G and to R. Details follow.

The pseudorandom generator $G=G_g$. Given as input $x\in\{0,1\}^n$ and parameter η, the algorithm G computes $g(x)\in\{0,1\}^k$ in time $k\cdot T'(n)$, and applies the generator $G^{\mathtt{IW}}$ from Theorem 6.3 with the constants α',β',η. The generator runs in time $\mathrm{poly}(k)$ and outputs a set of $\mathrm{poly}(k)$ strings $z_1,...,z_{\mathrm{poly}(k)}\in\{0,1\}^{k^\eta}$.

The reconstruction algorithm R. Given input $x\in\{0,1\}^n$ and parameter $\eta>0$, the algorithm R runs the reconstruction algorithm $R^{\mathtt{IW}}$ from Theorem 6.3 with input $(1^k,\eta)$. The reconstruction algorithm needs oracle access to a $(1/k^\eta)$-distinguisher for $G_g(x,\eta)$, which we provide using D_x; and it needs oracle access to the function whose truth-table is $g(x)$, which we provide using the T'-time algorithm for the output bits of g. The running time of $R^{\mathtt{IW}}$ is $O(k^{\beta'})$, and therefore the number of queries to D_x is at most $O(k^{\beta'})$ and the running time of this step is $O(k^{\beta'}\cdot T')$.

With probability at least $1-2^{-2k^\eta}$, at this point we have a circuit C_i such that $C_i^{D_x}$ correctly computes at least $1-\alpha'$ of the coordinates of $g(x)$. We evaluate C_i on all of its k inputs, answering its queries with D_x, and output the truth-table of C_i. The running time of this step is $\tilde{O}(k^{1+\beta'})$, and thus the total running time is $O(k^{\beta'}\cdot T)+\tilde{O}(k^{1+\beta'})$. ■

The following hardness-to-randomness result is a direct consequence of the reconstructive targeted PRG from Proposition 6.2. Loosely speaking, for a time function $T(n)$ and denoting $T'=T\cdot n^\epsilon$, the hypothesis below is that one-way functions exist, and there exists a function $g\colon\{0,1\}^n\to\{0,1\}^k$ such that the mapping $(x,i)\mapsto g(x)_i$ is computable in time T', but $g(x)$ cannot be approximately-printed with small error in time $T'\cdot n^\epsilon$, with high probability over x

chosen from a distribution $\mathbf{x}$. Given this hypothesis, the result asserts that we can derandomize probabilistic time T with essentially no overhead n^{ϵ}, on average with respect to the distribution $\mathbf{x}$.

Theorem 6.4 (superfast non-black-box derandomization from a non-batch-computable function)**.** *Let $T\colon \mathbb{N} \to \mathbb{N}$ be a polynomial, let $\delta'\colon \mathbb{N} \to (0,1)$, and let $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ be a polynomial-time-samplable ensemble of distributions, where $\mathbf{x}_n$ is over $\{0,1\}^n$. Assume that one-way functions exist, and that for every $\epsilon' > 0$ there exist $\alpha, \beta \in (0,1)$ and a function g mapping n bits to $n^{\epsilon'}$ bits such that:*

1. *There exists an algorithm that on input $(x,i) \in \{0,1\}^n \times [n^{\epsilon'}]$ outputs the i^{th} bit of $g(x)$ in time $T(n)\cdot n^{\epsilon'}$.*
2. *For every probabilistic algorithm A that runs in time $T(n)\cdot n^{(1+\beta)\cdot\epsilon'}$ and sufficiently large $n \in \mathbb{N}$, the probability over $x \sim \mathbf{x}_n$ that A approximately-prints $g(x)$ with error α is at most $\delta'(n)$.*

Then, for every $\epsilon > 0$ there exists a deterministic algorithm $D = D_{\epsilon}$ that runs in time $n^{\epsilon}\cdot T(n)$ and prints $t < n^{\epsilon}$ strings $w_1, ..., w_t \in \{0,1\}^{T(n)}$ such that the following holds. For every probabilistic machine M that runs in time T, with probability at least $1 - \delta'(n) - n^{-\omega(1)}$ over $x \sim \mathbf{x}_n$ it holds that

$$\left| \Pr_{r\in\{0,1\}^{T(n)}}[M(x,r)=1] - \Pr_{i\in[t]}[M(x,w_i)=1] \right| < n^{-\gamma} , \tag{6.1}$$

where $\gamma = \gamma_{\epsilon,\beta} > 0$ is a sufficiently small constant. Moreover, the algorithm D does not depend on the distribution $\mathbf{x}$ or on the parameter δ' from the hardness hypotheses.

The algorithm in the conclusion of Theorem 6.4 is similar to a targeted PRG (as in Definition 3.3), but it only works on average-case over a choice of $x \sim \mathbf{x}_n$ rather than for every x. This "targeted PRG" has seed length $\log(t) < \epsilon \cdot \log(n)$, and we will use it to deduce average-case derandomization with a multiplicative time overhead of n^{ϵ}.

Proof of Theorem 6.4. Let $\epsilon' = \epsilon/c$ for a sufficiently large constant $c > 1$ and let $k(n) = n^{\epsilon'}$. Let g be the corresponding function from our hypothesis, and note that with our new notation:

1. Each output bit of g is computable in time $T' = T \cdot k$.
2. For every probabilistic algorithm A running in time $T \cdot k^{1+\beta}$, the probability over $x \sim \mathbf{x}_n$ that A approximately-prints $g(x)$ with error α is at most δ'.

The deterministic algorithm D. Fix a sufficiently small constant $\mu = \mu_{\epsilon,\beta} > 0$. By Theorem 3.4, there exists a neg-PRG $G^{\texttt{crypto}}$ for polynomial-time algorithms that has stretch $n^{\mu} \mapsto T$ and running time $T(n)\cdot n^{\mu}$. We will use the algorithm $G = G_g$ from Proposition 6.2, with parameters $\alpha' = \alpha$ and $\beta' = \beta/2$ and η such that $k^{\eta} = n^{\mu}$ (i.e., $\eta = \mu/\epsilon'$); note that if μ is sufficiently small then η is sufficiently small. On input $x \in \{0,1\}^n$, our algorithm first computes the output-set of $G_g(x,\eta)$, which consists of poly(k) strings $z_1, ..., z_{\text{poly}(k)}$. Then, for each z_i it prints the string $w_i = G^{\texttt{crypto}}(z_i) \in \{0,1\}^{T(n)}$.

Note that the number of strings that D prints is poly$(k) < n^{\epsilon}$ (relying on a sufficiently large choice of c) and that the total running time of D is at most

$$k \cdot T'(n) + \text{poly}(k) \cdot T(n) \cdot n^{\mu} < n^{\epsilon} \cdot T(n) .$$

Also note that the algorithm D does not depend on $\mathbf{x}$ or on δ', but only on the hard function g and on the parameters ϵ, α, β.

Proof of correctness. Let M be a probabilistic machine that runs in time $T(n)$, and let γ be any constant smaller than μ. For a fixed input $x \in \{0,1\}^n$, denote by $\mathbf{z}_x$ the uniform distribution over $z_1, ..., z_{\text{poly}(k)}$. Note that for the fixed x, if Eq. (6.1) does not hold, then it cannot be that

$$M(x, \mathbf{u}_T) \approx_{1/n} M(x, G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) \approx_{n^{-\mu}} M(x, G^{\texttt{crypto}}(\mathbf{z}_x)) \ , \tag{6.2}$$

where $\approx_\alpha$ means that two distributions are α-close in statistical distance. (This is because the distribution $M(x, G^{\texttt{crypto}}(\mathbf{z}_x))$ in the RHS of Eq. (6.2) equals the distribution of $M(x, w_i)$ for a uniform $i \in [n^\epsilon]$, by the definition of the w_i's and of $\mathbf{z}_x$.)

We first claim that the probability over $x \sim \mathbf{x}_n$ that $M(x, \mathbf{u}_T) \not\approx_{1/n} M(x, G^{\texttt{crypto}}(\mathbf{u}_{n^\mu}))$ is $n^{-\omega(1)}$; that is:

Claim 6.4.1. *Let $S_n = \{x \in \{0,1\}^n : M(x, \mathbf{u}_T)$ is $(1/n)$-far from $M(x, G^{\texttt{crypto}}(\mathbf{u}_{n^\mu}))\}$. Then,* $\Pr[\mathbf{x}_n \in S_n] \leq n^{-\omega(1)}$.

Proof. The proof amounts to constructing a polynomial-time distinguisher for $G^{\texttt{crypto}}$ under the assumption $\Pr[\mathbf{x}_n \in S_n] \geq 1/\text{poly}(n)$. The crucial point in the construction is that the distinguisher can sample $x \sim \mathbf{x}_n$ and simulate M in polynomial time (since $\mathbf{x}_n$ is polynomial-time samplable and M runs in polynomial time). Details follow.

Assume that $\Pr[\mathbf{x}_n \in S_n] \geq 1/p(n)$ for some polynomial p. For every $x \in \{0,1\}^n$, denote $u(x) = \Pr[M(x, \mathbf{u}_T) = 1]$ and $w(x) = \Pr[M(x, G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1]$. Our distinguisher F for $G^{\texttt{crypto}}$ gets input $r \in \{0,1\}^T$, samples $x \sim \mathbf{x}_n$, estimates the values $u(x)$ and $w(x)$ each up to error $1/(4n \cdot p(n))$ and with confidence $1 - 2^{-2n}$, obtaining estimates $\widetilde{u}(x)$ and $\widetilde{w}(x)$ respectively, and outputs $F(r) = \begin{cases} M(x,r) & \widetilde{u}(x) > \widetilde{w}(x) \\ 1 - M(x,r) & \widetilde{u}(x) < \widetilde{w}(x) \end{cases}$.

Note that F runs in polynomial time, since T is a polynomial and $\mathbf{x}$ is polynomial-time-samplable and p is a polynomial. To see that F is an $n^{-O(1)}$-distinguisher for $G^{\texttt{crypto}}$, let $\texttt{FAR} = \left\{x : \left|u(x) - w(x)\right| > 1/(2n \cdot p(n))\right\}$, and note that $S_n \subseteq \texttt{FAR}$. We further partition $\texttt{FAR}$ into $\texttt{FAR}^{(>)} = \{x \in \texttt{FAR} : u(x) > w(x)\}$ and $\texttt{FAR}^{(<)} = \{x \in \texttt{FAR} : u(x) < w(x)\}$. Denote the random variable representing the coins that F uses for estimation by $\mathbf{v}$, and note that with probability at least $1 - 2^{-n}$ over $v \sim \mathbf{v}$ the following event, denoted by $\mathcal{V}$, happens: Conditioned on any choice of $x \in \texttt{FAR}^{(>)}$ we have that $F(r) = M(x,r)$, and conditioned on any choice of $x \in \texttt{FAR}^{(<)}$ we have that $F(r) = 1 - M(x,r)$. Then,

$$\begin{aligned}
&\Pr[F(\mathbf{u}_T) = 1] - \Pr[F(G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1] \\
&\quad = \mathbb{E}_{v\sim\mathbf{v},x\sim\mathbf{x}_n}\left[\Pr[F(\mathbf{u}_T) = 1|v,x] - \Pr[F(G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1|v,x]\right] \\
&\quad \geq \mathbb{E}_{x\sim\mathbf{x}_n}\left[\Pr[F(\mathbf{u}_T) = 1|\mathcal{V},x] - \Pr[F(G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1|\mathcal{V},x]\right] - 2^{-n} \\
&\quad \geq \Pr\left[\mathbf{x}_n \in \texttt{FAR}^{(>)}\right] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[\Pr[F(\mathbf{u}_T) = 1] - \Pr[F(G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1]|x \in \texttt{FAR}^{(>)}, \mathcal{V}\right] \\
&\qquad + \Pr\left[\mathbf{x}_n \in \texttt{FAR}^{(<)}\right] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[\Pr[F(\mathbf{u}_T) = 1] - \Pr[F(G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1]|x \in \texttt{FAR}^{(<)}, \mathcal{V}\right] \\
&\qquad + \Pr[\mathbf{x}_n \notin \texttt{FAR}] \cdot (-1/(2n\cdot p(n))) - 2^{-n} \\
&\quad = \Pr\left[\mathbf{x}_n \in \texttt{FAR}^{(>)}\right] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[\Pr[M(x,\mathbf{u}_T) = 1] - \Pr[M(x,G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 1]|x \in \texttt{FAR}^{(>)}, \mathcal{V}\right] \\
&\qquad + \Pr\left[\mathbf{x}_n \in \texttt{FAR}^{(<)}\right] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[\Pr[M(x,\mathbf{u}_T) = 0] - \Pr[M(x,G^{\texttt{crypto}}(\mathbf{u}_{n^\mu})) = 0]|x \in \texttt{FAR}^{(<)}, \mathcal{V}\right] \\
&\qquad - 1/(2n\cdot p(n)) - 2^{-n} \\
&\quad > \Pr\left[\mathbf{x}_n \in \texttt{FAR}^{(>)}\right] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[u(x) - w(x)|x \in \texttt{FAR}^{(>)}, \mathcal{V}\right] \\
&\qquad + \Pr\left[\mathbf{x}_n \in \texttt{FAR}^{(<)}\right] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[(1-u(x)) - (1-w(x))|x \in \texttt{FAR}^{(<)}, \mathcal{V}\right] \\
&\qquad - 1/(2n\cdot p(n)) - 2^{-n} \\
&\quad \geq \Pr[\mathbf{x}_n \in \texttt{FAR}] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[\left|u(x) - w(x)\right||x \in \texttt{FAR}, \mathcal{V}\right] - 1/p(n)^2 \\
&\quad \geq \Pr[\mathbf{x}_n \in S_n] \cdot \mathbb{E}_{x\sim\mathbf{x}_n}\left[\left|u(x) - w(x)\right||x \in S_n, \mathcal{V}\right] - 1/(2n\cdot p(n)) - 2^{-n} \\
&\quad \geq 1/(n\cdot p(n)) - 1/(2n\cdot p(n)) - 2^{-n}\ ,
\end{aligned}$$

where the last inequality is since $\Pr[\mathbf{x}_n \in S_n] \geq 1/p(n)$ and for every $x \in S_n$ it holds that $\left|u(x) - w(x)\right| > 1/n$. Thus, for a sufficiently large n the algorithm F is a $(1/(3n\cdot p(n)))$-distinguisher for $G^{\texttt{crypto}}$, a contradiction. □

Now, let $D_x(r) = M(x, G^{\texttt{crypto}}(r))$, and observe that for every $x \in \{0,1\}^n$ such that $M(x, G^{\texttt{crypto}}(\mathbf{u}_{n^\mu}))$ is $(n^{-\mu})$-far from $M(x, G^{\texttt{crypto}}(\mathbf{z}_x))$ it holds that D_x is a $(n^{-\mu})$-distinguisher for $\mathbf{z}_x$. We will now construct a probabilistic algorithm A that runs in time $k^{1+\beta}\cdot T(n)$ and approximately-prints $g(x)$ with error α for any such x. By our assumption, the probability over $x \sim \mathbf{x}_n$ that A approximately-prints $g(x)$ with error α is at most δ'. Thus, by a union bound, we will conclude that the probability over $x \sim \mathbf{x}_n$ that Eq. (6.2) does not hold, which upper-bound the probability over $x \sim \mathbf{x}_n$ that our derandomization errs on x, is at most $\delta' + n^{-\omega(1)}$.

Our algorithm A runs the reconstruction algorithm R from Proposition 6.2 with inputs (x, η), and while answering its oracles queries to D_x by simulating $G^{\texttt{crypto}}$ and M. The reconstruction R requires a $(k^{-\eta})$-distinguisher, and recall that $k^\eta = n^\mu$ by our parameter choices. The total running-time of A is

$$\tilde{O}(k^{1+\beta/2}) + k^{\beta/2}\cdot T' + \tilde{O}(k^{1+\beta/2})\cdot(T\cdot n^\mu) = o\Big(k^{1+\beta}\cdot T(n)\Big)\ ,$$

where we relied on the fact that R makes $\tilde{O}(k^{1+\beta/2})$ oracle queries, and that we can answer each oracle query in time $O(T \cdot n^{\mu})$. With probability at least $1 - 2^{-k^{\mu}} > 1 - \alpha$ it outputs a string that agrees with $g(x)$ on at least $1 - \alpha$ of the coordinates. ■

6.2 Proofs of Theorems 1.6 and 1.7 and of Corollary 1.8

In this section we show that Theorems 1.6 and 1.7 follow as corollaries of the hardness-to-randomness tradeoff in Theorem 6.4. Let us first formally state Theorem 1.7 and prove it:

Theorem 6.5 (superfast non-black-box derandomization over all polynomial-time-samplable distributions; Theorem 1.7, restated)**.** *Let $T\colon \mathbb{N} \to \mathbb{N}$ be a polynomial. Assume that one-way functions exist, and that for every $\epsilon > 0$ there exist $\delta > 0$ and a function g mapping n bits to n^{ϵ} bits such that:*

1. *There exists an algorithm that gets input $(x, i) \in \{0,1\}^n \times [n^{\epsilon}]$ and outputs the i^{th} bit of $g(x)$ in time $T'(n) = T(n) \cdot n^{\epsilon}$.*

2. *For every probabilistic algorithm A that runs in time $T'(n) \cdot n^{\delta}$ and every polynomial-time samplable distribution $\mathbf{x}$ and every sufficiently large n, the probability over $x \sim \mathbf{x}_n$ that A approximately-prints $g(x)$ with error .01 is negligible.*

Then $\mathcal{BPTIME}[T] \subseteq \bigcap_{\epsilon>0} \mathsf{heur}_{1-\mathsf{neg}}\text{-}\mathcal{DTIME}[n^{\epsilon} \cdot T]$, where neg is a negligible function.

Proof. Let $L \in \mathcal{BPTIME}[T]$, let M be a probabilistic machine that decides L in time T, and let $\epsilon > 0$. We invoke Theorem 6.4 with δ' that is a negligible function, with $\alpha = .01$ and $\beta = \delta/\epsilon$, and with *every* polynomial-time-samplable distribution $\mathbf{x}$. We obtain a deterministic algorithm D_{ϵ} printing strings that satisfy Eq. (6.1), and by the "moreover" part, the algorithm D_{ϵ} does not depend on $\mathbf{x}$, so we obtain the same algorithm D_{ϵ} for all choices of $\mathbf{x}$.

Our deterministic simulation A_L gets input $x \in \{0,1\}^n$ and outputs the majority value of $M(x, w_i)$ over all w_i's that D_{ϵ} prints. Its running time is $O(n^{\epsilon} \cdot T(n))$, since D_{ϵ} runs in time $n^{\epsilon} \cdot T(n)$ and prints n^{ϵ} strings, and its correctness follows by the conclusion of Theorem 6.4. ■

Remark 6.6. A non-black-box derandomization approach (such as our targeted PRGs) is *necessary* to obtain derandomization as in the conclusion of Theorem 6.4; that is, to obtain derandomization with a multiplicative overhead of less than n that works over arbitrary polynomial-time samplable distributions. To see this, fix any HSG G that supposedly derandomizes $pr\mathcal{BPTIME}[T]$ on average using $n^{1-\epsilon}$ seeds (i.e., seed length $(1-\epsilon) \cdot \log(n)$) and running time T'. Then, for a suitable problem in $pr\mathcal{BPTIME}[T]$, we can construct in deterministic time $O(n \cdot T')$ an input on which the corresponding derandomization errs. Specifically, we enumerate the outputs of G and construct a circuit that rejects these strings and accepts all other strings; this circuit is an input for $\mathsf{CAPP} \in pr\mathcal{BPTIME}[n] \subseteq pr\mathcal{BPTIME}[T]$ on which the derandomization errs.[29]

Since in Theorem 6.4 we construct a targeted PRG rather than just an algorithm that approximates the acceptance probability of a circuit, we are also able to deterministically *find* good random strings for probabilistic machines.[30] As stated in Corollary 1.8, this implies in particular

[29]With some care, one can strengthen this impossibility argument so that it holds also for $\mathcal{BPTIME}[T]$ rather than just $pr\mathcal{BPTIME}[T]$ (to do so we replace CAPP with PIT and construct an arithmetic circuit rather than a Boolean one).

[30]There is a known search-to-decision reduction in this context, first pointed out by Goldreich [Gol11a], but a-priori this reduction has a polynomial runtime overhead. See further discussion after the proof of Theorem 6.8.

that, under a hypothesis as in Theorem 6.5, the complexity of explicitly constructing combinatorial objects is nearly identical to the complexity of verifying that the combinatorial object meets the required specification. Let us formally define this notion and prove Corollary 1.8:

Definition 6.7 (explicit constructions). *Let $\Pi \subseteq \{0,1\}^*$ such that for every sufficiently large $n \in \mathbb{N}$ it holds that $\Pi \cap \{0,1\}^n \neq \emptyset$. We say that a deterministic algorithm A is an* explicit constrution *of Π if for every sufficiently large n, when A gets input 1^n it outputs an n-bit string in Π.*

Theorem 6.8 (explicit constructions in time that nearly matches the verification time; Corollary 1.8, restated). *Let $\Pi \in \mathcal{DTIME}[T]$ such that for every sufficiently large $n \in \mathbb{N}$ it holds that $|\Pi_n| \geq 2^n/n^{o(1)}$, where $\Pi_n = \Pi \cap \{0,1\}^n$. Then, under the assumption of Theorem 6.5, for every $\epsilon > 0$ there exists an explicit construction of Π in time $n^\epsilon \cdot T(n)$. Moreover, if the assumption of Theorem 6.5 holds for* every *polynomial T, then for every polynomial T we can deduce the conclusion while only assuming that $\Pi \in \mathcal{BPTIME}[T]$.*

Proof. Given $\epsilon > 0$, we invoke Theorem 6.4 with parameters δ', α, β as in the proof of Theorem 6.5 and with $\mathbf{x}$ such that $\mathbf{x}_n$ is supported only on the string 1^n (this is a polynomial-time-samplable distribution). Let $D_{T,\epsilon}$ be the deterministic algorithm from the conclusion of Theorem 6.4.

Let R_Π be a probabilistic machine that gets input $x \in \{0,1\}^n$, randomly chooses $\pi \in \{0,1\}^n$ and accepts if and only if $\pi \in \Pi$ (regardless of x). Note that R_Π runs in time $O(T(n))$. Our explicit construction of Π gets input 1^n, enumerates over the strings $w_1, ..., w_{n^\epsilon}$ that $D_{T,\epsilon}(1^n)$ prints, truncates each string to n bits (recall that the original strings are of length $T(n)$), and outputs the lexicographically-first truncated string r such that $R_\Pi(1^n, r) = 1$.

The foregoing algorithm runs in time $O(n^\epsilon \cdot T(n))$. To see that it is an explicit construction of Π, note that by Theorem 6.4, the probability over $x \sim \mathbf{x}_n$ that Eq. (6.1) is violated is less than 1; since $\mathbf{x}_n$ is just supported on 1^n, we deduce that Eq. (6.1) is satisfied for the machine R_Π above with input 1^n. Also, by our hypothesis that $|\Pi_n| \geq 2^n/n^{o(1)}$ and our definition of R_Π, we have that $\Pr[R_\Pi(1^n, \mathbf{u}_n) = 1] \geq n^{-o(1)}$. Hence, there always exists w_i such that $M(1^n, w_i) = 1$, which means that our algorithm indeed outputs some $\pi \in \Pi$.

For the "moreover" part, we cannot just define R_Π as above since we do not assume that $\Pi \in \mathcal{DTIME}[T]$. However, we assumed that there exists a probabilistic machine M that decides Π in time T. We define D_Π to be the machine that, given input $\pi \in \{0,1\}^n$, outputs the majority vote of $M(\pi, w_i)$ over the strings $w_1, ..., w_{n^\epsilon}$ that $D_{T,\epsilon}(\pi)$ outputs. Note that D_Π runs in time $T'(n) = O(n^\epsilon \cdot T(n))$ and that no probabilistic polynomial-time algorithm can find a string π such that $D_\Pi(\pi) \neq \Pi(\pi)$, with non-negligible probability.

We now define R_Π as above, using D_Π as the deterministic decision algorithm for Π (i.e., R_Π ignores its input $x \in \{0,1\}^n$, draws a random $\pi \in \{0,1\}^n$, and accepts iff $D_\Pi(\pi) = 1$). Since R_Π now runs in time $T'(n)$, we invoke Theorem 6.4 with time bound T' and with the same $\delta', \alpha, \beta, \mathbf{x}$ to obtain a deterministic algorithm $D_{T',\epsilon}$. Our explicit construction works just as above, only using the output strings of $D_{T',\epsilon}$ (instead of $D_{T,\epsilon}$) and evaluating them with the new R_Π that uses D_Π. To see that it is an explicit construction, the precise same analysis as above implies that our algorithm outputs π such that $D_\Pi(\pi) = 1$. Since this is a deterministic polynomial-time algorithm, by the properties of D_Π it cannot be that $\pi \notin \Pi$. ■

Remark 6.9. Under the particular hypotheses of Theorem 6.8 a different proof approach might be possible. Moreover, this alternative potential approach is more general, and does not need to assume that the superfast derandomization involves targeted PRGs (i.e., it only uses the generic derandomization $\mathcal{BPTIME}[T] \subseteq \bigcap_{\epsilon>0} \mathsf{heur}_{1-\mathsf{neg}}\text{-}\mathcal{DTIME}[n^{\epsilon} \cdot T]$). Specifically, when we assume one-way functions (as in Theorem 6.8), we can rely on the fast PRG in Theorem 3.4 to reduce the search-space of any explicit construction algorithm A from $\{0,1\}^n$ to $\{0,1\}^{n^{\epsilon}}$, for an arbitrarily small $\epsilon > 0$, without significantly affecting its running time or the relative density of valid outputs.[31] The next step is to apply a search-to-decision reduction as in [Gol11a], relying on the fact that the number of iterations is only n^{ϵ} for an arbitrarily small $\epsilon > 0$, and that it is infeasible to find in polynomial time an input on which the derandomization fails. While this proof approach might work, we prefer the approach presented in Theorem 6.8, both due to its simplicity and since it demonstrates the general point that targeted PRGs can be used directly to solve search problems (regardless of whether or not OWFs exist).

Let us now turn to proving Theorem 1.6. Recall that the result relies on a direct product hypothesis. To introduce this hypothesis we first establish what can be *unconditionally proved*, as a basis for comparison. For a function $f\colon \{0,1\}^n \to \{0,1\}$, denote by $f^{\times k}$ the k-wise direct-product $f^k(x_1, ..., x_k) = (f(x_1), ..., f(x_k)) \in \{0,1\}^k$. As mentioned in Section 1.2, it is *unconditionally* true that if a function f is average-case hard to compute in probabilistic time T, then $f^{\times k}(x)$ cannot be *approximately-printed* in similar time $\Omega(T)$ for the vast majority of inputs x.

To be more accurate, we will need a slightly stronger assumption, which was also needed in previous works that studied uniform direct-product results (see, e.g., [IJK+10], following [STV01; TV07]). Specifically, we will assume that hardness holds with respect to probabilistic algorithms that get a constant number of advice bits that depend on the *randomness* of the algorithm but still do not depend on the specific *input* to the algorithm. The class of problems solvable by probabilistic algorithms running in time T with a bits of randomness-dependent advice is denoted by $\mathcal{BPTIME}[T]//a$ (see [IJK+10] for further details and explanations). Then:

Proposition 6.10 (non-strong approximate-direct-product theorem)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $\delta \in (0, 1/2)$, let $\alpha > 0$ be sufficiently small, let $T\colon \mathbb{N} \to \mathbb{N}$ be time-computable, and let $k(n) = o(n)$. Then, for any $f \notin \mathsf{avg}_{1-\delta}\text{-}\mathcal{BPTIME}[T]//(1/\alpha)$ and any probabilistic algorithm A that runs in time $T(n) - n^c$, the probability over $z \in \{0,1\}^{k \cdot n}$ that A approximately-prints $f^{\times k}(z)$ with error α is at most δ.*

Proposition 6.10 follows as a corollary of the uniform list-decoding algorithms of Impagliazzo *et al.* [IJK+10] for the direct-product code (in particular, of [IJK+10, Theorem 1.8]). Since their paper uses slightly different notions (i.e., it refers to a direct-product function $f^{\times k}(x_1, ..., x_k)$ that only takes *distinct* inputs $x_1, ..., x_k$, and disallows $x_i = x_j$ for any $i \neq j \in [k]$), we include for completeness a proof of Proposition 6.10 in Appendix C.

The following hypothesis essentially says that Proposition 6.10 can be strengthened, for *some* function that is hard for probabilistic time T, to assert that the hardness of $f^{\times k}$ holds not only

[31]This is the case since we can replace the algorithm D_Π that decides Π by an algorithm D'_Π that gets as input a seed $s \in \{0,1\}^{n^{\epsilon}}$, expands it to an n-bit string using the PRG from Theorem 3.4, and accepts iff D_Π accepts. Thus, instead of finding a good object in $\{0,1\}^n$ (i.e., a string that D_Π accepts), it now suffices to find a good seed in $\{0,1\}^{n^{\epsilon}}$ (i.e., a string that D'_Π accepts).

with respect to algorithms running in time $T(n) - n^c$, but also with respect to algorithms running in time $T(n) \cdot k^{\beta}$, for some (arbitrarily small) constant $\beta > 0$. Specifically, the definition below is a more general version of the assumption that was stated in Section 1.2.

Assumption 6.11 (mildly-strong approximate-direct-product hypothesis)**.** *For any $\delta > 0$, the* mildly-strong approximate-direct-product hypothesis for time $T(n)$ and arity $k(n)$ and average-case success δ *asserts that there exist $\alpha, \beta > 0$ and a function $f \in \mathcal{DTIME}[T]$ such that the following holds. For any probabilistic algorithm A that runs in time $O(k^{\beta} \cdot T)$, the probability over $z \in \{0,1\}^{k \cdot n}$ that A approximately-prints $f^{\times k}(z)$ with error α is at most δ.*

Now we prove Theorem 6.12: Assuming that the mildly-strong approximate-direct-product hypothesis holds, and that one-way functions exist, probabilistic algorithms running in time T can be deterministically simulated with essentially no overhead, i.e. in time $T \cdot n^{\epsilon}$, on average over the uniform distribution.

Theorem 6.12 (superfast non-black-box derandomization from mildly-strong approximate-direct-product; Theorem 1.6, restated)**.** *Let $T \colon \mathbb{N} \to \mathbb{N}$ be a polynomial. Assume that there exist one-way functions secure against polynomial-time algorithms, and that for some $\delta > 0$ and every $\epsilon_0, \epsilon_1 > 0$ the mildly-strong approximate-direct-product hypothesis holds for time $T(n^{1-\epsilon_0}) \cdot n^{\epsilon_1}$ and arity n^{ϵ_1} and average-case success δ. Then $\mathcal{BPTIME}[T] \subseteq \bigcap_{\epsilon > 0} \mathsf{avg}_{1-\delta'}\text{-}\mathcal{DTIME}[n^{\epsilon} \cdot T]$, where $\delta'(n) = \delta(n) + n^{-\omega(1)}$.*

Proof. We show that for any $\epsilon' > 0$ there exist $\alpha, \beta > 0$ and a function g as in the hypothesis of Theorem 6.4 with $\mathbf{x}$ being the uniform distribution (i.e., for every $n \in \mathbb{N}$ it holds that $\mathbf{x}_n \equiv \mathbf{u}_n$). To see this, fix any $\epsilon' > 0$, and for $m(n) = n^{1/(1-\epsilon')}$ let $T' \colon \mathbb{N} \to \mathbb{N}$ such that $T'(n) = T(m(n)) \cdot m(n)^{\epsilon'}$ (i.e., $T'(n) = T(n^{1/(1-\epsilon')}) \cdot n^{\epsilon'/(1-\epsilon')}$). Let $f \in \mathcal{DTIME}[T']$ be a function satisfying the mildly-strong approximate-direct-product hypothesis for time T' and arity $k(n) = m(n)^{\epsilon'} = n^{\epsilon'/(1-\epsilon')}$ and average-case success δ. By our hypothesis, the output bits of the function $g = f^{\times k}$ can be computed in time $T'(n) = T(m) \cdot m^{\epsilon'}$, whereas there exist $\alpha, \beta > 0$ such that $g(z)$ cannot be approximately-printed with error α on more than δ of the inputs z in probabilistic time $T'(n) \cdot m(n)^{\beta} = T(m(n)) \cdot m(n)^{(1+\beta) \cdot \epsilon'}$.

Let $\epsilon > 0$, let D_{ϵ} be the deterministic algorithm from Theorem 6.4, and for every $x \in \{0,1\}^n$ let $w_1(x), ..., w_{n^{\epsilon}}(x)$ be the strings that $D_{\epsilon}(x)$ prints. For any $L \in \mathcal{BPTIME}[T]$ and $\epsilon > 0$, our deterministic algorithm outputs the majority value of $M_L(x, w_i(x))$ over all $i \in [n^{\epsilon}]$, where M_L is a probabilistic T-time machine that decides L. This algorithm runs in time $O(n^{\epsilon} \cdot T(n))$ and correctly decides L with probability at least $1 - \delta'$ over choice of $x \in \{0,1\}^n$. ■

6.3 Necessity of non-batch-computable functions

We now show one natural setting in which superfast derandomization necessitates the existence of a multi-output function $g \colon \{0,1\}^n \to \{0,1\}^{n^{\epsilon}}$ whose individual bits can be computed in time T, but that cannot be computed in time significantly less than $T \cdot n^{\epsilon}$.

The computational model is that of balanced formulas of fan-in two over the full binary basis that are highly uniform (see the precise uniformity condition below). The size of a formula is the number of leaves, and is typically denoted by T. We assume for simplicity that the size is always a power of two, that internal gates are over the full binary basis, and that negations appear only in the bottom level (i.e., we only negate variables).

Probabilistic uniform formulas can be defined in several ways, since randomness can be used either by the machine that constructs the formula or by the formula itself (or by both); moreover, when randomness is used by the formula itself, it can be either read-once or reusable.[32] Our result works in any of these models, and in none of the models does there seem to be a trivial derandomization with the requirements that are specified below (i.e., it does not seem trivial to simulate highly-uniform probabilistic formulas by slightly larger highly-uniform deterministic formulas). For concreteness, we define probabilistic formulas as ones in which the formula uses randomness in a read-once manner: That is, some leaves are labeled by a special symbol R, which indicates that the leaf tosses an independent random coin (and the probabilistic computation of the formula on a fixed input is over a uniform distribution for the values of leaves that are labeled by R).

Fixing a canonical indexing of the $[2T-1]$ gates in a balanced formula of size T such that the leaves are indexed by $[T]$, the formula is uniquely determined by the labels of the internal gates (i.e., the function that they compute) and by the labels of the leaves (i.e., each leaf is labeled by a literal, or by a constant in $\{0,1\}$, or by R). In particular, such a formula is uniquely described by its label function:

Definition 6.13 (label function)**.** *For a formula F_n of size T over n variables, the* labels function of *F_n, denoted $\Phi_F\colon [2T-1] \to [2n] \cup \{\mathsf{T},\mathsf{F},\mathsf{R}\} \cup \mathfrak{B}_2$, maps each gate in F_n to its label, where a label in $[2n]$ is interpreted as a literal, the labels* T *and* F *stand for the constants* 1 *and* 0 *respectively, the label* R *indicates that this is a leaf gate that tosses a random coin, and $\mathfrak{B}_2$ is the full binary basis (i.e., the set of all functions $\{0,1\}^2 \to \{0,1\}$).*

The uniformity condition that we impose on the formulas is similar to what is known as DPOLYLOGTIME uniformity. Specifically, we require that the label function of a size-T formula can be computed in time $\text{polylog}(T)$ (i.e., polynomial in the input length to the label function).

Definition 6.14 (well-structured formulas)**.** *For $c \geq 1$, we say that an n-bit formula F of size T is* c-well-structured *if F is balanced and its label function Φ_F is computable in time $\log(T)^c$.*

The following statement asserts that superfast derandomization of well-structured formulas implies the existence of a "non-batch-computable" function as we wanted.

Proposition 6.15 (necessity of non-batch-computable functions for derandomization of well-structured formulas)**.** *Let c be any sufficiently large constant, let $c_1 > c$, let $\epsilon, \delta \in (0,1)$ such that $\epsilon > \delta$, and let $T\colon \mathbb{N} \to \mathbb{N}$ be a polynomial.*

- **Assumption:** *Assume that for every polynomial $T_0\colon \mathbb{N} \to \mathbb{N}$, every function computable by a c-well-structured family of probabilistic formulas of size T_0 is also computable, for every sufficiently large $n \in \mathbb{N}$ and on more than 0.99 of the inputs $x \in \{0,1\}^n$, by a c_1-well-structured family of deterministic formulas of size $T_0(n) \cdot n^{\delta}$.*

[32]That is, in the former case leaves that are random bits are independent, and in the latter case the same random bit can reappear as the label of several leafs.

- **Conclusion:** *Then, there exists $g\colon \{0,1\}^n \to \{0,1\}^{n^\epsilon}$ such that the mapping $(x,i) \mapsto g(x)_i$ is computable in time $\tilde{O}(T)$, but the following holds. For any c-well-structured family $\{D_n\}$ of probabilistic formulas of size $o(T(n) \cdot n^{\epsilon-\delta})$ there exists infinitely many input lengths $n \in \mathbb{N}$ such that the fraction of $x \in \{0,1\}^n$ for which* $\Pr[D_n(x) = g(x)] \geq 2/3$ *is less than* 0.01.

High-level proof idea. Let us explain the proof idea before presenting the full proof. For $T' = T \cdot n^\epsilon$, consider well-structured formulas of size T' whose top part computes the parity function of the gates at depth $k = \epsilon \cdot \log(n)$. We first diagonalize against this class using a function $L^{\mathtt{diag}}$ that is itself computable by a well-structued formula family $\left\{F_n^{\mathtt{diag}}\right\}_{n\in\mathbb{N}}$ of the same size T'. Specifically, for every $n \in \mathbb{N}$, let F_n be the well-structured formula that is described by the uniform machine whose index is n. [33] The diagonalizing formula $F_n^{\mathtt{diag}}$ has the precise same structure as F_n, the only difference being that exactly one sub-formula in level k computes the negation of the corresponding sub-formula in F_n. This is indeed a diagonalization, since whenever the top $k-1$ levels compute the parity function we have that $F_n^{\mathtt{diag}}(x) \neq F_n(x)$ for all inputs $x \in \{0,1\}^n$.

Now, the hard function g gets input $x \in \{0,1\}^n$ and outputs the values of gates in level k of $F_n^{\mathtt{diag}}(x)$. Observe that each output bit of g is computable in time $\tilde{O}(T)$, since the formulas for $L^{\mathtt{diag}}$ are well-structured (i.e., balanced and with an efficiently-computable label function), which means that each output bit corresponds to a subformula of size $T'/n^\epsilon = T$. To see that g is hard, assume that it can be computed by a family of well-structured probabilistic formulas $\{D_n\}$ of size $T_0 = o(T')$ that is "too small". By our assumption, it can also be computed by a family $\{D'_n\}$ of well-structured deterministic formulas, which (for simplicity) we now assume are of precisely the same size T_0. We now define a well-structured formula family $\{D''_n\}$ that first uses D'_n to compute g, and then computes the parity of the outcomes. For every n that is associated with the uniform machine that defines D''_n, on the one hand we have that D''_n computes the same function as $F_n^{\mathtt{diag}}$ (since both of them compute the parity of the values of g), but on the other hand D''_n is of size $o(T')$ – a contradiction to the diagonalizing properties of $F^{\mathtt{diag}}$.

We are left with just an issue, which is that our derandomization does not succeed in worst-case, but only on $\mu > 0.99$ of inputs. To deal with this issue, recall that the function $\left\{F_n^{\mathtt{diag}}\right\}$ diagonalizes against every M on *all* inputs of the corresponding length n. Thus, it suffices to assume that the derandomization of the probabilistic formulas of size T_0 that compute g succeeds even just on *one input* on which the foregoing formulas correctly compute g.[34]

Proof of Proposition 6.15. Let $c_2 > c_1$ be a sufficiently large constant, and let $T'(n) = T(n) \cdot n^\epsilon$. Fix a standard representation of Turing machines by integers such that each machine is associated with infinitely many input lengths, and for $n \in \mathbb{N}$, let M_n be the machine associated with n.

The hard function. We will not need to formally define $F_n^{\mathtt{diag}}$ that was mentioned above, and instead we just directly define $g\colon \{0,1\}^n \to \{0,1\}^{n^\epsilon}$, by the following algorithm. On input

[33] We may not assume that every machine describes a well-structured formula, but we can nevertheless correct non-valid outputs in some canonical way.

[34] This follows since the "towards-a-contradiction" hypothesis is that the probabilistic formulas succeed on 0.01 of the inputs, and the derandomization hypothesis is that the deterministic simulation succeeds on $\mu > 0.99$ of the inputs.

$x \in \{0,1\}^n$, we think of each $i \in [n^\epsilon]$ as the index of a gate v_i in the k^{th} level of a balanced formula on n variables of size $T'(n)$, where k is the level that is $\epsilon \cdot \log(n)$ below the output gate.[35] Since we consider a balanced formula, the subformula rooted at v_i is of size $T'(n)/2^k = T(n)$. We run M_n on all inputs j such that j is the index of a gate in the subtree of v_i, each time for at most $\log(T'(n))^{c_2}$ steps, and correct the outputs of M_n if they are not valid in some fixed canonical way.[36] Then, we define $g(x)_1 = 1 - v_1(x)$ and for $i \in \{2, ..., n^\epsilon\}$ we define $g(x)_i = v_i(x)$.

By the definition of g, the mapping $(x,i) \mapsto g(x)_i$ can be computed in time $\tilde{O}(T(n))$. Now, denote by F_n the formula that is described by the M_n (after correcting invalid outputs in the canonical way above) and for $x \in \{0,1\}^n$ denote by $F_{n,k}(x)$ the values of the gates in the k^{th} level of F_n at input x. Then, we have that

$$\oplus_i F_{n,k}(x)_i = 1 - \oplus_i g(x)_i \ . \tag{6.3}$$

Analysis. Assume towards a contradiction that there is a family $\{D_n\}$ of c-well-structured probabilistic formulas of size $T_0(n) = o(T'(n)/n^\delta)$ such that for every sufficiently large $n \in \mathbb{N}$, the fraction of inputs $x \in \{0,1\}^n$ such that $\Pr[D_n(x) = g(x)] \geq 2/3$ is at least 0.01. By our assumption, there is a family $\{D'_n\}$ of c_1-well-structured deterministic formulas of size $T_0(n) \cdot n^\delta = o(T'(n))$ such that for every sufficiently large $n \in \mathbb{N}$ there exists $x_n \in \{0,1\}^n$ for which $D'_n(x) = g(x)$. Let $M^{(1)}$ be the uniform machine describing $\{D'_n\}$.

For $T_1(n) = T_0(n) \cdot n^\delta < T'(n)$, consider the following machine $M^{(2)}$. Given $i \in [2T_1(n) - 1]$, if i is an index of a gate in the k^{th} level of a formula of size $T_1(n)$ or in a level beneath it, then $M^{(2)}(i) = M^{(1)}(i)$; and otherwise (if i is the index of a gate in the top $k-1$ levels), then $M^{(2)}(i)$ outputs the parity function. Finally let $M^{(3)}$ be a machine with essentially the same functionality as $M^{(2)}$, the only difference being that $M^{(3)}$ describes a "padded" formula of size $T'(n)$ with only $T_1(n)$ non-trivial gates at its top, and a trivial copying mechanism at its bottom levels (in order for the formula to be of size precisely $T'(n)$).[37]

Note that for all $n \in \mathbb{N}$ and every input in $[2T'(n) - 1]$ it holds that $M^{(3)}$ runs in time at most $\log(T_0(n) \cdot n^\delta)^{c_1} + \log^2(n) \leq \log(T'(n))^{c_2}$, where $c_2 = c_1 + 2$ and the additional term of $\log^2(n)$ accounts for the overheads of $M^{(3)}$ on top of $M^{(1)}$ (i.e., deciding whether the input requires simulating $M^{(1)}$, or outputting the parity function, or implementing a trivial copying mechanism). Thus, $M^{(3)}$ describes a c_2-well-structured formula family $\{D''_n\}$ of size T'. Denoting by $D''_{n,k}(x)$ the values of gates in the k^{th} level of D''_n at input x, we have that

$$D''_n(x) = \oplus_i D''_{n,k}(x)_i = \oplus_i D'_n(x)_i \ . \tag{6.4}$$

Now, fix any $n \in \mathbb{N}$ such that $M^{(3)}$ is associated with input length n. Since $M^{(3)}$ describes a c_2-well-structured family and by the definition of F_n (when describing the hard function g above), for any such n we have that $F_n = D''_n$ and $F_{n,k} = D''_{n,k}$. Also, by our assumption, there exists x_n

[35] Recall that we fixed a canonical ordering of the gates in a balanced formula. This canonical ordering induces an ordering on the gates in the k^{th} level, and i is the index of v_i in that ordering.

[36] That is, if $M_n(u) \notin \mathfrak{B}_2$ for an internal node u we let $M_n(u) = \wedge$, and if $M_n(u) \notin [2n] \cup \{\mathsf{T}, \mathsf{F}\}$ for a leaf u then $M_n(u) = 1$, representing x_1.

[37] That is, $M^{(3)}(i)$ behaves like $M^{(2)}(i)$ on the first $T_1(n) - 1$ gates. Then the next $T_1(n)$ gates, which are the leaves of the formula described by $M^{(2)}$ and are denoted by $\mathcal{L}$, are $\vee$ gates. Similarly all other gates except for the leaves are $\vee$ gates. Finally $M^{(3)}$ labels each leaf ℓ by the label that $M^{(2)}$ assigns to the unique node in $\mathcal{L}$ that is an ancestor of ℓ.

such that $D'_n(x_n) = g(x_n)$. Hence, by combining Eqs. (6.3) and (6.4) we have that

$$\oplus_i g(x_n)_i = 1 - \oplus_i F_{n,k}(x_n)_i = 1 - \oplus_i D''_{n,k}(i) = 1 - \oplus_i D'_n(x)_i = 1 - \oplus_i g(x)_i \ ,$$

a contradiction. ■

We note several shortcomings of Proposition 6.15. First, the function g is computable in time $\tilde{O}(T)$ but not necessarily by well-structured formulas of such size. Secondly, our assumption referred to superfast derandomization of search problems, rather than only decision problems. And thirdly, we only deduced that $g(x)$ cannot be printed, whereas in Theorems 1.6 and 1.7 we needed a lower bound even on printing a "slightly-corrupted" version of $g(x)$.

7 Open problems

We conclude this paper with several intriguing open questions stemming from our results.

1. (**Direct construction of targeted PRGs.**) In Section 4 we constructed targeted HSGs from instance-wise hardness (Proposition 4.5), and the immediate open problem is to strengthen this construction to yield targeted PRGs. The lack of targeted PRGs forces us to rely on the inclusion $pr\mathcal{BPP} \subseteq pr\mathcal{RP}^{pr\mathcal{RP}}$ to get the implication "almost-all-input hardness implies $pr\mathcal{BPP} = pr\mathcal{P}$". This indirect approach causes the hardness-randomness tradeoff in Theorem 1.3 to be not as smooth as the tradeoff for $pr\mathcal{RP}$, which also prevents us to get a full equivalence (between hardness assumption and derandomization conclusion) in the "low-end" setting (see Theorem 5.17). Thus, getting a similar construction of targeted PRGs would significantly improve the tradeoff for $pr\mathcal{BPP}$, as well as strengthen Theorem 5.17 into a full equivalence.

   The core technical challenge can be abstracted out as follows (see also Section 4.4): Given polynomially many *candidate* targeted PRGs for a given randomized algorithm on a particular input (meaning, at least one of the candidates is a valid targeted PRG on this particular input, and others may behave arbitrarily), can we still derandomize the relevant probabilistic algorithm (that has a two-sided error) on this input[38] A similar issue also occurred in previous works concerning uniform hardness-to-randomness (see e.g., [CRT+20]).

2. (**Relaxing the low-depth requirement on the hard function.**) In Theorem 1.2 we were able to deduce $pr\mathcal{BPP} = pr\mathcal{P}$ from almost-all-input hardness of function f computable by *low-depth* circuits. The ultimate goal would be to achieve the same conclusion with almost-all-input hardness of an arbitrary polynomial-time function, thus showing a complete equivalence between almost-all-input hardness and derandomization. One intermediate question is whether or not we can relax the constraint on the hard function f from low-depth to low-space; that is, can we show $pr\mathcal{BPP} = pr\mathcal{P}$ follows from the existence of a function $f : \{0,1\}^n \to \{0,1\}^n$ computable by algorithms running in polynomial time and

[38]To get derandomization with one-sided error one can simply enumerate all the outputs of the targeted PRGs and see if one of them leads the probabilistic algorithm to accept the given input.

in space (say) n^2 such that f is almost-all-input hard for probabilistic algorithms running in fixed polynomial time n^c (i.e., when $c \gg 2$ is some unknown fixed constant)?

Since our work crucially relies on the doubly-efficient proof systems for low-depth circuits by [GKR15], it is natural to suspect the recent doubly-efficient proof systems for low-space computation by [RRR16] may be helpful for the above intermediate question. We nevertheless note that our *particular* techniques cannot be directly applied to the proof system of [RRR16], since bootstrapping systems exist only for functions computable by bounded-depth circuits (see Proposition 4.2).

3. (**The needed assumptions for superfast derandomization.**) In Theorem 1.7 we show that $\mathcal{BPTIME}[T] \subseteq \text{heur-}\mathcal{DTIME}[T \cdot n^{\epsilon}]$ under the existence of OWFs against uniform adversaries and non-batch-computable functions. In Proposition 6.15 we show that the assumption about non-batch-computable functions is *necessary* in the setting of derandomizing DPOLYLOGTIME-uniform balanced formulas, but the general case remains unclear. It is therefore important to understand whether one can further weaken the two assumptions in Theorem 1.7 (e.g., can we get rid of the assumption about OWFs?), or show these assumptions are indeed necessary. Another potential approach to this question would be to discover different assumptions under which one can deduce superfast derandomization.

4. (**Application to the $\mathcal{BPL} = \mathcal{L}$ question.**) In this paper we demonstrate that almost-all-input hardness is tightly related to the $pr\mathcal{BPP} = pr\mathcal{P}$ question. Can our framework be applied to the $\mathcal{BPL} = \mathcal{L}$ question as well? Perhaps our framework can inspire new construction of targeted PRGs for derandomizing bounded-space computations (e.g., showing $\mathcal{BPL} = \mathcal{L}$)? So far, most of the work in this direction focused on black-box derandomization of $\mathcal{BPL}$.[39]

5. (**Proving almost-all-input lower bounds against uniform algorithms.**) Can we prove the hardness assumption in Theorem 1.2? It would immediately imply $pr\mathcal{BPP} = pr\mathcal{P}$. Perhaps a first step is to construct a polynomial-time function $f\colon \{0,1\}^n \to \{0,1\}^n$ that cannot be computed by polynomial-size uniform probabilistic $\mathcal{AC}^0$ circuits on *almost all inputs*.

Acknowledgements

We are very grateful to Madhu Sudan, who suggested the proof idea underlying Appendix B (the latter is crucially used in our results concerning logspace-uniform $\mathcal{NC}$). We thank Ryan Williams (who is the PhD advisor of the first author and the postdoc host of the second author) for very valuable discussions and perspectives, and in particular for suggesting to prove tighter results for $\mathcal{NC}$ circuits, and for pointing out the relevance of Kozen's work and of the "simulation versus diagonalization" question. We are grateful to Avishay Tal for a helpful conversation and for the useful suggestion to randomly choose a small set S in the proof of Proposition 6.10. We also thank Salil Vadhan for a very useful conversation, in particular about transformations of bootstrapping systems into low-depth circuits. We thank Oded Goldreich and Igor Oliveira for detailed comments and suggestions on an early version of the paper, which significantly

[39]One notable exception is the seminal work by Reingold [Rei08] proving that undirected connectivity is in $\mathcal{L}$.

improved our exposition. In particular, we thank Igor for the suggestion to point out implications of our derandomization for search problems, which led to stating and proving Corollary 1.8.

References

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

[ACR98] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. "A new general derandomization method". In: *Journal of the ACM* 45.1 (1998), pp. 179–213.

[Alm19] Josh Alman. "An illuminating algorithm for the light bulb problem". In: *Proc. 2nd Symposium on Simplicity in Algorithms (SOSA)*. 2019, Art. No. 2, 11.

[BF99] Harry Buhrman and Lance Fortnow. "One-Sided Versus Two-Sided Error in Probabilistic Computation". In: *Proc. 16th Symposium on Theoretical Aspects of Computer Science (STACS)*. 1999, pp. 100–109.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. "Non-Deterministic Exponential Time has Two-Prover Interactive Protocols". In: *Comput. Complex.* 1 (1991), pp. 3–40.

[BFN+93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. "BPP has subexponential time simulations unless EXPTIME has publishable proofs". In: *Computational Complexity* 3.4 (1993), pp. 307–318.

[BGH82] Allan Borodin, Joachim von zur Gathen, and John E. Hopcroft. "Fast Parallel Matrix and GCD Computations". In: *Inf. Control.* 52.3 (1982), pp. 241–256. DOI: 10.1016/S0019-9958(82)90766-5. URL: https://doi.org/10.1016/S0019-9958(82)90766-5.

[BM84] Manuel Blum and Silvio Micali. "How to Generate Cryptographically Strong Sequences of Pseudo-random Bits". In: *SIAM Journal of Computing* 13.4 (1984), pp. 850–864.

[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.

[CIS18] Marco L. Carmosino, Russell Impagliazzo, and Manuel Sabin. "Fine-grained derandomization: from problem-centric to resource-centric complexity". In: *Proc. 45th International Colloquium on Automata, Languages and Programming (ICALP)*. 2018, Art. No. 27, 16.

[CLW20] Lijie Chen, Xin Lyu, and Richard Ryan Williams. "Almost-Everywhere Circuit Lower Bounds from Non-Trivial Derandomization". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020.

[CNS99] Jin-Yi Cai, Ajay Nerurkar, and D. Sivakumar. "Hardness and hierarchy theorems for probabilistic quasi-polynomial time". In: *Proc. 31st Annual ACM Symposium on Theory of Computing (STOC))*. 1999, pp. 726–735.

[CR20] Lijie Chen and Hanlin Ren. "Strong average-case lower bounds from non-trivial derandomization". In: *Proc. 52th Annual ACM Symposium on Theory of Computing (STOC)*. 2020, pp. 1327–1334.

[CRT+20] Lijie Chen, Ron D. Rothblum, Roei Tell, and Eylon Yogev. "On Exponential-Time Hypotheses, Derandomization, and Circuit Lower Bounds". In: *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2020, pp. 13–23.

[Csa76] L. Csanky. "Fast Parallel Matrix Inversion Algorithms". In: *SIAM J. Comput.* 5.4 (1976), pp. 618–623. URL: https://doi.org/10.1137/0205040.

[CT21] 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.

[CW19] Lijie Chen and R. Ryan Williams. "Stronger Connections Between Circuit Analysis and Circuit Lower Bounds, via PCPs of Proximity". In: *Proc. 34th Annual IEEE Conference on Computational Complexity (CCC)*. 2019, 19:1–19:43.

[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.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for muggles". In: *Journal of the ACM* 62.4 (2015), Art. 27, 64.

[GL89] Oded Goldreich and Leonid A. Levin. "A Hard-core Predicate for All One-way Functions". In: *Proc. 21st Annual ACM Symposium on Theory of Computing (STOC)*. 1989, pp. 25–32.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001. ISBN: 0-521-79172-3. DOI: 10.1017/CBO9780511546891.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. New York, NY, USA: Cambridge University Press, 2008.

[Gol11a] Oded Goldreich. "In a World of P=BPP". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*. 2011, pp. 191–232.

[Gol11b] Oded Goldreich. "Two Comments on Targeted Canonical Derandomizers". In: *Electronic Colloquium on Computational Complexity: ECCC* 18 (2011), p. 47.

[Gol18] Oded Goldreich. "On doubly-efficient interactive proof systems". In: *Foundations and Trends® in Theoretical Computer Science* 13.3 (2018), front matter, 1–89.

[GR17] Oded Goldreich and Guy N. Rothblum. "Worst-case to Average-case reductions for subclasses of P". In: *Electronic Colloquium on Computational Complexity: ECCC* 26 (2017), p. 130.

[GSTS03] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. "Uniform hardness versus randomness tradeoffs for Arthur-Merlin games". In: *Computational Complexity* 12.3-4 (2003), pp. 85–130.

[GV08] Dan Gutfreund and Salil Vadhan. “Limitations of hardness vs. randomness under uniform reductions”. In: *Proc. 12th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 2008, pp. 469–482.

[GVW11] Oded Goldreich, Salil Vadhan, and Avi Wigderson. “Simplified derandomization of BPP using a hitting set generator”. In: *Studies in complexity and cryptography*. Vol. 6650. Lecture Notes in Computer Science. Springer, Heidelberg, 2011, pp. 59–67.

[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: *J. Comput. Syst. Sci.* 65.4 (2002), pp. 695–716.

[HIL+99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. “A Pseudorandom Generator from any One-way Function”. In: *SIAM Journal of Computing* 28.4 (1999), pp. 1364–1396.

[Hoz19] William M. Hoza. “Typically-correct derandomization for small time and space”. In: *Proc. 34th Annual IEEE Conference on Computational Complexity (CCC)*. 2019, Art. No. 9, 39.

[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.

[IJK+10] Russell Impagliazzo, Ragesh Jaiswal, Valentine Kabanets, and Avi Wigderson. “Uniform direct product theorems: simplified, optimized, and derandomized”. In: *SIAM Journal of Computing* 39.4 (2010), pp. 1637–1665.

[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.

[IW98] R. Impagliazzo and A. Wigderson. “Randomness vs. Time: De-Randomization Under a Uniform Assumption”. In: *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1998, pp. 734–.

[IW99] Russell Impagliazzo and Avi Wigderson. “P = BPP if E requires exponential circuits: derandomizing the XOR lemma”. In: *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*. 1999, pp. 220–229.

[Kab01] Valentine Kabanets. “Easiness assumptions and hardness tests: trading time for zero error”. In: vol. 63. 2. 2001, pp. 236–252.

[KM98] Adam Klivans and Dieter van Melkebeek. “Graph Nonisomorphism has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses”. In: *Electronic Colloquium on Computational Complexity: ECCC* 5 (1998), p. 75.

[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.

[Koz78] Dexter Kozen. “Indexing of subrecursive classes”. In: *Proc. 10th Annual ACM Symposium on Theory of Computing (STOC)*. 1978, pp. 287–295.

[Koz92] Dexter Campbell Kozen. *Design and Analysis of Algorithms*. Texts and Monographs in Computer Science. Springer, 1992. ISBN: 978-3-540-97687-5. DOI: 10.1007/978-1-4612-4400-4. URL: https://doi.org/10.1007/978-1-4612-4400-4.

[Lau83] Clemens Lautemann. “BPP and the polynomial hierarchy”. In: *Information Processing Letters* 17.4 (1983), pp. 215–217.

[LOS21] Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. “Pseudodeterministic Algorithms and the Structure of Probabilistic Time”. In: *Proc. 53rd Annual ACM Symposium on Theory of Computing (STOC)*. 2021.

[LP21] Yanyi Liu and Rafael Pass. “On One-way Functions from NP-Complete Problems”. In: *Electronic Colloquium on Computational Complexity: ECCC* 28 (2021), p. 059.

[Lu01] Chi-Jen Lu. “Derandomizing Arthur-Merlin games under uniform assumptions”. In: *Computational Complexity* 10.3 (2001), pp. 247–259.

[MS05] Dieter van Melkebeek and Rahul Santhanam. “Holographic proofs and derandomization”. In: *SIAM Journal of Computing* 35.1 (2005), pp. 59–90.

[MW18] Cody Murray and Ryan Williams. “Circuit Lower Bounds for Nondeterministic Quasi-Polytime: An Easy Witness Lemma for NP and NQP”. In: *Proc. 50th Annual ACM Symposium on Theory of Computing (STOC)*. 2018.

[NIR03] Alan Nash, Russell Impagliazzo, and Jeff Remmel. “Universal Languages and the Power of Diagonalization”. In: *Proc. 18th Annual IEEE Conference on Computational Complexity (CCC)*. 2003, p. 337.

[NIR06] Alan Nash, Russell Impagliazzo, and Jeff Remmel. “Infinitely-Often Universal Languages and Diagonalization”. In: *Electronic Colloquium on Computational Complexity: ECCC* 13 (2006), p. 51.

[Nis91] Noam Nisan. “Pseudorandom bits for constant depth circuits”. In: *Combinatorica* 11.1 (1991), pp. 63–70.

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

[OL21] Igor Carboni Oliveira and Zhenjian Lu. “An efficient coding theorem via probabilistic representations and its applications”. In: *Proc. 48th International Colloquium on Automata, Languages and Programming (ICALP)*. 2021.

[Oli19] Igor Carboni Oliveira. “Randomness and Intractability in Kolmogorov Complexity”. In: *Proc. 46th International Colloquium on Automata, Languages and Programming (ICALP)*. 2019, 32:1–32:14.

[Rei08] Omer Reingold. “Undirected connectivity in log-space”. In: *J. ACM* 55.4 (2008), 17:1–17:24.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-round interactive proofs for delegating computation". In: *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC)*. 2016, pp. 49–62.

[Sap17] Ramprasad Saptharishi. *Algebra and Computation*. 2017. URL: https://www.tifr.res.in/~ramprasad.saptharishi/assets/courses/2017-algComp/algComp_2017.pdf.

[Sha03] Ronen Shaltiel. "Towards proving strong direct product theorems". In: *Computational Complexity* 12.1-2 (2003), pp. 1–22.

[Sha11] Ronen Shaltiel. "Weak derandomization of weak algorithms: explicit versions of Yao's lemma". In: *Computational Complexity* 20.1 (2011), pp. 87–143.

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

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. "Pseudorandom generators without the XOR lemma". In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 236–266.

[SU05] Ronen Shaltiel and Christopher Umans. "Simple extractors for all min-entropies and a new pseudorandom generator". In: *Journal of the ACM* 52.2 (2005), pp. 172–216.

[SU07] Ronen Shaltiel and Christopher Umans. "Low-end uniform hardness vs. randomness tradeoffs for AM". In: *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*. 2007, pp. 430–439.

[Sud15] Madhu Sudan. *Algebra and Computation, Lecture 9*. 2015. URL: http://people.seas.harvard.edu/~madhusudan/MIT/ST15/scribe/lect09-madhu.pdf.

[Sud21] Madhu Sudan. *Personal Communication*. 2021.

[Sud97] Madhu Sudan. "Decoding of Reed Solomon Codes beyond the Error-Correction Bound". In: *J. Complex.* 13.1 (1997), pp. 180–193. DOI: 10.1006/jcom.1997.0439. URL: https://doi.org/10.1006/jcom.1997.0439.

[SW13] Rahul Santhanam and Ryan Williams. "On medium-uniformity and circuit lower bounds". In: *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*. 2013, pp. 15–23.

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

[TV07] Luca Trevisan and Salil P. Vadhan. "Pseudorandomness and Average-Case Complexity Via Uniform Reductions". In: *Computational Complexity* 16.4 (2007), pp. 331–364.

[Uma03] Christopher Umans. "Pseudo-random generators for all hardnesses". In: *Journal of Computer and System Sciences* 67.2 (2003), pp. 419–440.

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

[Wil13] Ryan Williams. "Improving Exhaustive Search Implies Superpolynomial Lower Bounds". In: *SIAM Journal of Computing* 42.3 (2013), pp. 1218–1244.

[Yao82] Andrew C. Yao. "Theory and Application of Trapdoor Functions". In: *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1982, pp. 80–91.

[Zim08] Marius Zimand. "Exposure-resilient extractors and the derandomization of probabilistic sublinear time". In: *Computational Complexity* 17.2 (2008), pp. 220–253.

Appendix A Instantiations of known PRG and code constructions

In this appendix we state and prove various instantiations of known constructions of PRGs and of codes. The point in stating and proving these is that we instantiate the known constructions with non-standard parameters and with specific efficiency constraints that are useful for our purposes. Specifically, the appendix includes proofs of three claims:

1. We prove Theorem 4.8, which is an instantiation of the Nisan-Wigderson PRG in which both the output length and the running time of the reconstruction algorithm are very small, and both the PRG and the reconstruction can be computed by $\mathcal{NC}$ circuits that are *logspace-uniform* (assuming that the PRG gets explicit access to the hard function, and that the reconstruction algorithm gets oracle access to a distinguisher).

2. We prove Theorem 6.3 from Section 6, which combines the foregoing construction with parts of the Impagliazzo-Wigderson [IW99] construction, and asserts that the reconstruction algorithm for the obtained PRG is a uniform learning algorithm (following [IW98]).

3. We prove Theorem 4.9, which asserts that the list-decoding algorithm of Goldreich and Levin [GL89] can be implemented by small-depth circuits.

In Section A.1 we prove Theorem 4.8. Then, in Section A.2 we state and prove an instantiation of the derandomized direct-product construction of [IW99], where the main point is that the reconstruction algorithm is uniform and efficient. In Section A.3 we combine the two foregoing constructions to prove Theorem 6.3. And in Section 4.9, which can be read independently of the rest of this appendix, we prove Theorem 4.9.

A.1 The Nisan-Wigderson PRG using logspace-uniform circuits

To compute the Nisan-Wigderson PRG by logspace-uniform $\mathcal{NC}$ circuits, we will use a logspace algorithm that constructs combinatorial designs and "hard-wires" the designs into an $\mathcal{NC}$ circuits. The logspace construction of designs appears in Section A.1.1, and the corresponding instantiation of the NW PRG appears in Section A.1.2.

A.1.1 Designs in logspace

The construction of combinatorial designs that we present works in space that is logarithmic in the number of sets. We follow an idea that appeared in [HR03] and was attributed to Salil Vadhan; an earlier construction appeared in [KM98]. Let us first recall the definition of an explicit standard combinatorial design, and then state and prove the result itself.

Definition A.1 ([NW94]). *A family of sets $S_1, \ldots, S_m \subseteq [d]$ is called an (m, ℓ, ρ, d)-design if each of the sets is of size $|S_i| = \ell$, and any two distinct sets S_i, S_j satisfy $|S_i \cap S_j| \leq \log(\rho)$. The computational complexity of the design is the complexity of the function that gets input $i \in [m]$ and outputs the set S_i.*

The proof idea is to create a designs by simulating a "code concatenation" procedure akin to the Justensen code. Loosely speaking, we combine a good and explicit "outer" error-correcting code with an optimal "inner" combinatorial design (that we can find by an exhaustive search). The result of this combination is stated in the following lemma from [HR03]:

Lemma A.2 (from codes to designs; see [HR03, Lemma 5.5]). *For $a, \ell_0, \rho_0, d_0, \ell_1, k, m \in \mathbb{N}$, assume that there exist:*

1. *An (a, ℓ_0, ρ_0, d_0)-design computable in space $O(\log(m))$.*
2. *An error-correcting code $C\colon [m] \to [a]^{\ell_1}$ with (absolute) distance $\ell_1 - k$ that is computable in space $O(\log(m))$.*

Then, there exists an (m, ℓ, ρ, d)-design computable in space $O(\log(m))$, where $\ell = \ell_1 \cdot \ell_0$ and $d = \ell_1 \cdot d_0$ and $\log(\rho) = k \cdot \ell_0 + (\ell_1 - k) \cdot \log(\rho_0)$.

We obtain the following logspace-computable designs by combining the Reed-Solomon code with an inner design that we find using exhaustive search.

Lemma A.3 (designs in logspace). *There exists a universal constant $c \geq 1$ such that the following holds. For any constant $\alpha \in (0, 1)$ and sufficiently large integer ℓ there exists an (m, ℓ, ρ, d)-design, where $m \geq 2^{(1/c)\cdot\alpha\cdot\ell}$ and $d \leq c \cdot \ell/\alpha$ and $\log(\rho) = \alpha \cdot \ell$, that is computable in space $O(\log(m))$.*

Proof. Let $\ell_1 = O(\ell/\log(\ell))$ be the minimal power of two such that $\ell' = \ell_1 \cdot \log(\ell_1) \geq \ell$. As the code we take the Reed-Solomon code $\mathbb{F}_{\ell_1}^{k+1} \to \mathbb{F}_{\ell_1}^{\ell_1}$ of degree $k = \lfloor \alpha/8 \rfloor \cdot \ell_1$, whose number of codewords is $m = \ell_1^{k+1} > 2^{(\alpha/8)\cdot\ell'}$. We will combine this code with a design of ℓ_1 sets of size $\ell_0 = \log(\ell_1)$ in a universe of size $d_0 = O(\ell_0/\alpha)$ that pairwise-intersect on $\log(\rho_0) = (\alpha/8) \cdot \ell_0$ coordinates (see, e.g., [Vad12, Problem 3.2] for the existence of such a design).

Using Lemma A.2, the resulting design has $m > 2^{(\alpha/8)\cdot\ell'}$ sets of size ℓ' in a universe of size $O(\ell'/\alpha)$ whose pairwise-intersections are of size at most

$$k \cdot \ell_0 + (\ell_1 - k) \cdot \log(\rho_0) \leq (\alpha/8) \cdot \ell' + (\alpha/8) \cdot \ell' - k \cdot \log(\rho_0) < \alpha/4 \cdot \ell' \,.$$

We truncate each set in the design to have exactly ℓ elements. Note that $\ell' < 4\ell$,[40] and therefore the pairwise-intersections are of size at most $\alpha \cdot \ell$ and the number of sets is $m \geq 2^{(\alpha/32)\cdot\ell}$.

Note that the Reed-Solomon code $\mathbb{F}_{\ell_1}^{\lfloor\alpha/8\rfloor\cdot\ell_1+1} \to \mathbb{F}_{\ell_1}^{\ell_1}$ can be computed in space $O(\ell_1) = O(\log(m))$. Also, we can find an optimal design of ℓ_1 sets of size $\log(\ell_1)$ in a universe of size $O(\log(\ell_1)/\alpha)$ by an exhaustive search, in space $O(\ell_1 \cdot \log(\ell_1)) = O(\log(m))$. By Lemma A.2, the space complexity of the final design is $O(\log(m))$. ■

[40]This is since $\ell_1 \cdot \log(\ell_1) < 4(\ell_1/2) \cdot \log(\ell_1/2)$ for a sufficiently large $\ell_1 = O(\ell/\log(\ell))$.

A.1.2 The NW PRG with uniform reconstruction

We now prove Theorem 4.8. As mentioned above, this is an instantiation of the NW PRG in which the output length is small, the reconstruction time is small, and both the PRG and the reconstruction algorithm are computable by logspace-uniform $\mathcal{NC}$ circuits (assuming they are given access to the hard function and to a distinguisher, respectively).

Theorem A.4 (the NW PRG with reconstruction as a learning algorithm)**.** *There exists a universal constant $c > 1$, an oracle machine G, and a probabilistic oracle machine R_0, such that the following holds:*

1. **Generator:** *When given input $(1^{\ell_k}, 1^m, \alpha)$ such that $m \leq 2^{(\alpha/c)\cdot\ell_k}$ oracle access to $h\colon \{0,1\}^{\ell_k} \to \{0,1\}$, the machine G runs in time $2^{c\cdot\ell_k/\alpha}$ and outputs a set of strings in $\{0,1\}^m$. Moreover, if α is constant and ℓ_k and m are sufficiently large, then G can be implemented by logspace-uniform oracle circuits of size $2^{c\cdot\ell_k/\alpha}$ and depth $O(\log(m, \ell_k))$.*

2. **Reconstruction:** *When given input $(1^{\ell_k}, 1^m, \alpha)$ and oracle access to a $(1/m)$-distinguisher D for $G^h(1^{\ell_k}, 1^m, \alpha)$ and to h, the machine R_0 runs in time $m^c \cdot 2^{\alpha\cdot\ell_k}$, makes non-adaptive queries, and outputs with probability at least $1 - 2^{-3m}$ an oracle circuit that computes h on $1/2 + m^{-3}$ of the inputs when given access to D. The circuit that R_0 outputs has depth $\mathrm{polylog}(m, \ell_k)$ and makes just one oracle query. Moreover, if α is a constant and ℓ_k and m are sufficiently large, then R_0 can be implemented by a logspace-uniform probabilistic oracle circuit of size $m^c \cdot 2^{\alpha\cdot\ell_k}$ and depth $\mathrm{polylog}(m, \ell_k)$ that makes non-adaptive queries.*

Proof. Given input $(1^{\ell_k}, 1^m)$ and access to $h \in \{0,1\}^{2^{\ell_k}}$, and assuming that c in our hypothesis is sufficiently large, the machine G constructs the following combinatorial design: The design consists of m sets $S_1, ..., S_m \subseteq [d]$ of size $|S_i| = \ell_k$ that pairwise-intersect on at most $\alpha \cdot \ell_k$ coordinates in a universe of size $d = O(\ell_k/\alpha)$. The machine then enumerates in parallel over seeds $z \in \{0,1\}^d$, and for every z it queries the oracle m times and outputs the m-bit string $h(z\restriction_{S_1}) \circ ... \circ h(z\restriction_{S_m})$ (where $z\restriction_{S_i}$ is the projection of z to the coordinates specified by S_i).

If α is constant and ℓ_k, m are sufficiently large, then the foregoing procedure can be implemented by a logspace-uniform circuit of size $2^d \cdot \mathrm{poly}(m) \leq 2^{c\cdot\ell_k/\alpha}$ and depth $O(\log(m, \ell_k))$, assuming again that c is sufficiently large. Specifically, to compute the designs by logspace-uniform circuits we use Lemma A.3; the logspace algorithm that constructs the circuit computes the designs in advance and hard-wires them into the circuit. In general (i.e., without assuming that α is a constant), the foregoing procedure can be implemented in time $2^{(c/\alpha)\cdot\ell_k}$, using a standard construction of designs in time $\mathrm{poly}(m)$ (see, e.g., [Vad12, Problem 3.2]).

The machine R_0 gets input $(1^{\ell_k}, 1^m, \alpha)$ and constructs the same design that G constructed. It then repeats the following expriment for $\mathrm{poly}(m)$ attempts in parallel:

1. Randomly choose an index $i \in [m]$, values $z' \in \{0,1\}^{[d]\setminus S_i}$, values $r_{i,...,m} \in \{0,1\}^{m-i+1}$, and a random bit σ. Query h on the $m \cdot 2^{\alpha\cdot\ell_k}$ points corresponding to intersections of S_i with other sets.

2. Create a circuit that gets input $x \in \{0,1\}^{\ell_k}$, combines x and z' to a seed $z \in \{0,1\}^d$ (by placing x in the positions indexed by S_i and using z' to fill the other positions), looks up $r_1, ..., r_{i-1} = h(z\restriction_{S_1}) \circ ... \circ h(z\restriction_{S_{i-1}})$, creates a string $r = r_1, ..., r_m$, makes one oracle call to the distinguisher D with input r, and outputs $D(r) \oplus \sigma$.

3. Use $\mathrm{poly}(m)$ oracle calls to h to test whether or not the circuit agrees with h on at least $1/2 + m^{-3}$ of the inputs, with confidence $1 - 2^{-4m}$.

After performing the $\mathrm{poly}(m)$ experiments, the machine R_0 checks if one of them succeeded, and if so it outputs the circuit produced by one of the successful experiments. Otherwise, the machine R_0 halts and outputs "fail".

By a standard reconstruction argument following [NW94], in any one of the experiments, with probability at least $1/O(m)$ the circuit correctly computes h on $1/2 + 1/O(m^2) > 1/2 + m^{-3}$ of the inputs. Thus, the probability that at least one of the $\mathrm{poly}(m)$ attempts will succeed is at least $1 - 2^{-3m}$ the machine R_0.

The procedure R_0 runs in time $\mathrm{poly}(m) \cdot 2^{\alpha \cdot \ell_k}$ and makes non-adaptive oracle queries. The circuit that R_0 prints has depth $\mathrm{polylog}(m, \ell_k)$ and makes just one oracle call to D. Moreover, if α is constant then R_0 can be implemented by a logspace-uniform circuit, since we compute the designs in logspace (as we did for M), the first two steps are computationally very simple, and in the third step we just need to simulate a logspace-uniform circuit (the one we constructed in the second step); the size of this circuit is $\mathrm{poly}(m) \cdot 2^{\alpha \cdot \ell_k}$, and its depth is $\mathrm{polylog}(m, \ell_k)$. ■

A.2 The derandomized direct product of IW with uniform reconstruction

We now state and prove an instantiation of the derandomized direct-product construction of Impagliazzo and Wigderson [IW99]. The point of the statement and proof below is simply to verify that the reconstruction algorithm for the [IW99] construction is both *uniform* and *very quick*.

Theorem A.5 (the locally list-decodable "derandomized direct-product" code of [IW99]). *For every two constants $\delta, \gamma > 0$, let $c = c_{\delta,\gamma} > 1$ be a sufficiently large constant. Then, for every sufficiently large k and $r \le \log(k)$ and $\eta = 2^{-r/c}$ there exists a mapping of $g \in \{0,1\}^k$ to $h \in \{0,1\}^{\mathrm{poly}(k)}$ that satisfies the following:*

1. **Low complexity overhead:** *There exists an algorithm that gets input $i \in [\mathrm{poly}(k)]$ and outputs the i^{th} entry of h in time $\mathrm{polylog}(k)$ with r oracle queries to g.*

2. **Reconstruction:** *There exists a probabilistic algorithm that gets input 1^k and oracle access to g, runs in time $k^{\gamma/2} \cdot \mathrm{poly}(\log(k), 1/\eta)$, and outputs $O(1/\eta^2)$ oracle circuits such that the following holds. For every function $\widetilde{h}$ that agrees with h on $1/2 + \eta$ of the inputs, with probability at least $1 - \exp(-1/\eta)$ one of the oracle circuits correctly computes g on $1 - \delta$ of the inputs with at most $\mathrm{poly}(\log(k)/\eta)$ queries to $\widetilde{h}$.*

Proof. To specify our construction we need to recall the following construction of a generator G_0 from [IW99] that maps a random seed of length k' (where the requirements on k' will be clarified below) to an output of length $r \cdot \log(k)$. The generator `XOR`s the output of two algorithms, which are applied independently to the same seed:

1. The first algorithm is a randomness-efficient sampler that maps its seed to r samples of $\log(k)$ bits such that any set in $\{0,1\}^{\log(k)}$ of density δ is sampled with accuracy $\delta/2$ and with confidence $\eta^4/9 = 2^{-\Omega(r)}$.

2. The second algorithm is a nearly-disjoint subsets generator (a-la [NW94]). This algorithm considers a family of subsets $S_1, ..., S_r \subseteq [k']$, each of length $\log(k)$, that pairwise intersect on at most $(\gamma/2) \cdot \log(k)$ coordinates. Given a seed z, the algorithm outputs the r substrings $(z\restriction_{S_i})_{i \in [r]}$ (where $z\restriction_{S_i}$ is the projection of z to the locations specified by S_i).

For a seed length k' that is the maximum among the seed lengths of the two algorithms above, let $h_{\mathsf{IW}} \colon \{0,1\}^{k'} \to \{0,1\}^r$ be the function $h_{\mathsf{IW}}(z) = (g(G_0(z)_1), ..., g(G_0(z)_r))$, where $G_0(z)_i$ is the i^{th} output string of G_0, which is of length $\log(k)$. The main lemma from [IW99] is the following reconstruction algorithm, which transforms a function that agrees with h_{IW} on $\mathrm{poly}(\eta)$ of the inputs to a function that agrees with g on almost all inputs:

Lemma A.5.1. *There exists a probabilistic oracle machine R_{IW} that, given access to g, produces in time $\tilde{O}(k^{\gamma/2}/\eta^9)$ a deterministic oracle circuit $\widetilde{g}$ such that with probability $1 - \exp(-1/\eta)/2$ the following holds: When given access to a function that agrees with h_{IW} on at least η^4 of the inputs, the circuit $\widetilde{g}$ makes $O(\log(k)/\eta^9)$ queries and correctly computes g on at least $1 - \delta$ of the inputs.*

Proof sketch. We follow the proof of Theorem 15 in [IW99] with parameters $\epsilon = \eta^4$ and $\rho = \delta/2$ and $q = \frac{\epsilon}{4\delta/\rho+1} = \eta^4/9 > 2^{-\rho \cdot r/6}$ (our sampler indeed satisfies the hypotheses of the theorem with these parameters[41]) and with $M = k^{\gamma/2}$ (our nearly-disjoint generator satisfies the hypotheses of the theorem with this parameter).

The proof in [IW99] describes a probabilistic algorithm R_{IW} that runs in time $\tilde{O}(k^{\gamma/2})$ and constructs a probabilistic oracle circuit C with one oracle gate that satisfies the following. For every function $\widetilde{h}_{\mathsf{IW}}$ that agrees with h_{IW} on ϵ of the inputs there exists a set $X \subset \{0,1\}^{\log(k)}$ of density at least $1 - \delta$ such that for every $x \in X$, the expectation (over random coins of R_{IW}) of the probability (over random coins of C) that the circuit with access to $\widetilde{h}_{\mathsf{IW}}$ outputs $h_{\mathsf{IW}}(x)$ is at least $1/2 + q$. We modify R_{IW} so that it chooses fixed random coins for C, in which case for every $x \in X$, with probability at least $1/2 + q$ over coins for R_{IW} the resulting deterministic oracle circuit outputs $h_{\mathsf{IW}}(x)$ when given access to $\widetilde{h}_{\mathsf{IW}}$.

By running R_{IW} for $t' = O(q^{-2} \cdot \log(k) \cdot (1/\eta)) = O(\log(k)/\eta^9)$ times and printing a circuit that outputs the majority decision of the t' circuits, with probability at least $1 - \exp(-1/\eta)/2$ the printed circuit computes h_{IW} for every $x \in X$, while making at most t' oracle calls to $\widetilde{h}_{\mathsf{IW}}$. $\square$

The mapping of g to h. We are given $g \in \{0,1\}^k$ and wish to map it to $h \in \{0,1\}^{\mathrm{poly}(k)}$. We instantiate the generator G_0 with the following two algorithms. For a sampler, we can use any strongly-explicit expander with constant spectral gap: The seed is thus of length $\log(k) + O(r)$ (specifying a vertex in $[k]$ and r edge indices), and we rely on an appropriate expander Chernoff bound (see, e.g., [AB09, Theorem 21.15]), using the fact that r is larger than a sufficiently large constant, to deduce that any set of density δ is sampled with accuracy $\delta/2$ and with confidence $2^{-r/10}$. For a nearly-disjoint subsets generator we use a standard construction of combinatorial designs in time $\mathrm{poly}(r, \log(k/\gamma))$ with universe size $O(\log(k/\gamma))$ (see, e.g., [Vad12, Problem 3.2]).

[41] In [IW99] the requirement from a sampler is stronger, and non-standard in modern terms: It specifies that any product of δ-dense sets $T_1 \times ... \times T_r \in (\{0,1\}^{\log(k)})^r$ should be sampled approximately well (see [IW99, Definition 5]). However, their proof only uses the property that is considered standard today, which refers to the special case in which there is a single δ-dense set $T \subseteq \{0,1\}^{\log(k)}$ and $T_i = T$ for all $i \in [r]$. Moreover, the confidence parameter of the sampler that we use is much better than what is needed in the [IW99] proof.

Combining these constructions yields seed length $k' = \max\{\log(k) + O(r), O(\log(k)/\gamma)\} = O(\log(k))$, and a generator computable in time $\mathrm{poly}(k')$.

Recall that $h_{\mathsf{IW}}(z) = (g(G_0(z)_1), ..., g(G_0(z)_r))$. We define h to be the truth-table of the Hadamard encoding of h_{IW}; that is, h is the truth-table of $h\colon \{0,1\}^{k'+r} \to \{0,1\}$ such that $h(z, \alpha) = \oplus_{i\in[r]}\alpha_i \cdot h_{\mathsf{IW}}(z)_i$. Note that the truth-table of h is of length $2^{k'+r} = \mathrm{poly}(k)$, and that we can compute each entry in this truth-table in time $\mathrm{polylog}(k)$ with r queries to g.

The reconstruction algorithm. Let $\widetilde{h}\colon \{0,1\}^{k'+r} \to \{0,1\}$ be a function that agrees with h on $1/2 + \eta$ of the inputs. The algorithm of [GL89] runs in time $\mathrm{poly}(\log(k)/\eta)$, and with probability at least $1 - \exp(-1/\eta)/2$ outputs an oracle circuit $\widetilde{h}_{\mathsf{IW},\mathtt{list}}$ that gets input $(z, i) \in \{0,1\}^{k'} \times [t]$, where $t = O(1/\eta^2)$, and satisfies the following: For at least $\Omega(\eta)$ of the inputs z to h_{IW} there exists $i \in [t]$ such that $\widetilde{h}^{\widetilde{h}}_{\mathsf{IW},\mathtt{list}}(z, i) = h_{\mathsf{IW}}(z)$ (see, e.g., [Gol08, Theorem 7.8][42]). When the algorithm of [GL89] succeeds, by an averaging argument there exists $i \in [t]$ such that the circuit $\widetilde{h}^{(i)}_{\mathsf{IW}}$ defined by $\widetilde{h}^{(i)}_{\mathsf{IW}}(z) = \widetilde{h}_{\mathsf{IW},\mathtt{list}}(z, i)$, when given oracle access to $\widetilde{h}$, agrees with h_{IW} on at least $\Omega(\eta/t) = \Omega(\eta^3) > \eta^4$ of the inputs.

We run the probabilistic oracle machine R_{IW} from Lemma A.5.1, while answering its queries using our oracle to g, and this machine outputs an oracle circuit $\widetilde{g}$. We output the t oracle circuits obtained by composing $\widetilde{g}$ with each of the circuits $\widetilde{h}^{(i)}_{\mathsf{IW}}$, for $i \in [t]$. The running time of our algorithm is $\tilde{O}(k^{\gamma/2}) \cdot \mathrm{poly}(1/\eta)$, and each of the t circuits makes at most $\mathrm{poly}(\log(k)/\eta)$ queries to $\widetilde{h}$. With probability at least $1 - \exp(-1/\eta)$, both algorithms (of [GL89] and of [IW99]) succeeded, in which case one of the t circuits computes g on $1 - \delta$ of the inputs with oracle access to $\widetilde{h}$. ■

A.3 The combined construction

We now state Theorem 6.3 and prove it. The proof amounts to a straightforward combination of Theorems A.4 and A.5.

Theorem A.6 (the PRG of [NW94; IW99] with reconstruction as a high-accuracy learning algorithm)**.** *For every two constants $\delta, \gamma > 0$ there exist an oracle machine G and a probabilistic oracle machine R such that for every function $g\colon \{0,1\}^{\log(k)} \to \{0,1\}$ and sufficiently small $\epsilon = \epsilon_{\delta,\gamma} > 0$ the following holds.*

- **Generator:** *When given input $(1^k, \epsilon)$ and oracle access to g, the machine G runs in time $\mathrm{poly}(k)$ and outputs a set of strings in $\{0,1\}^m$, where $m = k^\epsilon$.*
- **Reconstruction:** *When given input $(1^k, \epsilon)$ and oracle access to a $(1/m)$-distinguisher D for $G^g(1^k, \epsilon)$ and to g, the machine R runs in time $O(k^\gamma)$ and with probability at least $1 - 2^{-2m}$ outputs an oracle circuit that agrees with g on $1 - \delta$ of the inputs when given access to D.*

Proof. We instantiate Theorem A.5 with parameters γ and $\delta/2$. Let $r = 3c \cdot \log(m)$, where $c = c_{\delta/2,\gamma} > 1$ is the constant from Theorem A.5, and note that $r \le \log(k)$ by our assumption that

[42] The algorithm of [GL89] is probabilistic, and for every z in a set Z of density $\Omega(\eta)$, with probability at least $1 - \exp(-1/\eta^2)$ there exists i such that the i^{th} potential output of the algorithm equals the correct result. By choosing randomness and fixing it into this probabilistic algorithm, with probability at least $1 - \exp(-1/\eta)/2$ we obtain a circuit that succeeds (in the sense above) on at least half of the inputs in Z.

ϵ is sufficiently small, and that $\eta = 2^{-r/c} = 1/m^3$. Let $h\colon \{0,1\}^{\ell_k} \to \{0,1\}$ be the corresponding function from Theorem A.5, where $\ell_k = O(\log(k))$. The machine G is the one from Theorem A.4, instantiated with h and with the parameter m. Note that the running time of G is polynomial in $2^{\ell_k} = \mathrm{poly}(k)$.

The machine R gets oracle access to a distinguisher D and to g. It first runs the machine R_0 from Theorem A.4, obtaining an oracle circuit C'_0. (Note that R_0 requires oracle access to h, and R only has oracle access to g, but R can answer each query to h using $r \le \log(k)$ queries to g.) Then, R runs the reconstruction algorithm from Theorem A.5, which yields $t = O(1/\eta^2) = O(m^6)$ oracle circuits $C'_1, ..., C'_t$. It replaces the oracle gates in each C'_i with the circuit C'_0 (which contains an oracle gate to D), and outputs the t resulting circuits $C_1, ..., C_t$.

By Theorem A.4, with probability at least $1 - 2^{-3m}$ the function $(C'_0)^D$ agrees with h on $1/2 + m^{-3} = 1/2 + \eta$ of the inputs. Conditioned on this event, by Theorem A.5 with probability at least $1 - \exp(-1/\eta) > 1 - 2^{-3m}$ there exists $i \in [t]$ such that C_i^D agrees with g on $1 - \delta/2$ of the inputs. For $i \in [t]$, we estimate the fraction of inputs on which C_i^D agrees with g, up to error $\delta/2$ and with confidence $1 - 2^{-3m}/t$ (by randomly sampling $O(m)$ inputs, making oracle calls to g, and evaluating C_i with oracle calls to D). With probability at least $1 - 2^{-3m}$, after this step we find a single C_i such that C_i^D agrees with g on at least $1 - \delta$ of the inputs. The running time of R is $\mathrm{poly}(m) \cdot k^{\gamma/2} < k^\gamma$, and its error probability is at most $3 \cdot 2^{-3m} < 2^{-2m}$. ■

A.4 List-decoding the Hadamard code by small-depth circuits of small

We now state and prove Theorem 4.9, which asserts that the list-decoding algorithm of Goldreich and Levin [GL89] can be implemented by small-depth circuits. The proof amounts to verifying that the standard construction of [GL89] satisfies this property.

Theorem A.7 (list-decoding the Hadamard code [GL89]; Theorem 4.9, restated)**.** *For any time-computable $a\colon \mathbb{N} \to \mathbb{N}$ satisfying $a(\ell_0) \le \ell_0$ and $\epsilon\colon \mathbb{N} \to (0,1/2)$ there exists a transformation* $\mathtt{Had}$ *that maps any function $g\colon \{0,1\}^{\ell_0} \to \{0,1\}^{a(\ell_0)}$ to a Boolean function $\mathtt{Had}(g)\colon \{0,1\}^{\ell_0 + a(\ell_0)} \to \{0,1\}$ such that the following holds.*

1. **Encoding:** *For every $x \in \{0,1\}^{\ell_0}$ and $z \in \{0,1\}^{a(\ell_0)}$ it holds that $\mathtt{Had}(g)(x,z) = \langle g(x), z\rangle = \bigoplus_{i\in[a(\ell_0)]} g(x)_i \cdot z_i$.*
2. **Decoding:** *There exists a logspace-uniform circuit* GL *of size* $\mathrm{poly}(\ell_0/\epsilon)$ *and depth* $\mathrm{polylog}(\ell_0/\epsilon)$ *that gets input 1^{ℓ_0} and outputs a probabilistic oracle circuit C of depth* $\mathrm{polylog}(\ell_0/\epsilon)$ *that satisfies the following. For every oracle $\widetilde{\mathtt{Had}}(g)$ that agrees with $\mathtt{Had}(g)$ on $1/2 + \epsilon$ of the inputs, the probability over the random coins of C and a choice of $x \in \{0,1\}^{\ell_0}$ that $C^{\widetilde{\mathtt{Had}}(g)}(x) = g(x)$ is at least* $\mathrm{poly}(\epsilon)$.

Proof. For convenience we denote $n = \ell_0$ and $H = \widetilde{\mathtt{Had}}(g)$. For $k = O(\log(n/\epsilon))$, the probabilistic oracle circuit C circuit gets input $x \in \{0,1\}^n$ and chooses at random k bits $w_1, ..., w_k \in \{0,1\}$ and k vectors $s^1, ..., s^k \in \{0,1\}^{a(n)}$. For each $i \in [a(n)]$ in parallel, the circuit C computes $y_i = \mathtt{MAJ}\{b_{i,S}\}_{S\subseteq[k]}$, where $b_{i,S} = H(x, \sum_{j\in S} s^j + e_i) \oplus \sum_{j\in S} w_j$ and $e_i \in \{0,1\}^{a(n)}$ is the indicator vector of the i^{th} position. The circuit outputs the n-bit string $y_1, ..., y_n$. A standard analysis shows that the success probability of the resulting circuit is at least $\mathrm{poly}(\epsilon)$ (see, e.g., [AB09, Proof of Theorem 9.12]). The depth of C is $\mathrm{polylog}(n, 1/\epsilon)$, since for each output bit C only requires a constant number of iterated addition operations on $\mathrm{poly}(n/\epsilon)$ vectors of n bits. ■

Appendix B Algorithms in logspace-uniform $\mathcal{NC}$ for polynomial problems

Our goal in this appendix is to prove that low-degree polynomials are sample-aided worst-case to rare-case reducible by logspace-uniform $\mathcal{NC}$ circuits. To prove this we follow the known algorithms underlying such a result and show that each of these algorithms can be implemented by logspace-uniform $\mathcal{NC}$ circuits.

The main idea in the argument (which appears in Section B.2) was suggested to us by Madhu Sudan [Sud21], to whom we are very grateful. In high-level, we first show how to extract linear factors from bivariate polynomials, then use this to obtain a list-decoder for the Reed-Solomon (RS) code, then deduce a local list-decoder for the Reed-Muller (RM) code, and finally obtain the worst-case to rare-case reduction. In more detail:

1. First, in Section B.1, we recall some standard results asserting that logspace-uniform $\mathcal{NC}$ circuits can solve basic arithmetic and linear-algebraic problems.

2. In Section B.2 we show how, given a bivariate polynomial $p \in \mathbb{F}[x, y]$, we can find all of its linear factors of the form $y - p(x)$ in logspace-uniform $\mathcal{NC}$. This is the crux of the argument, which was suggested by Madhu Sudan.

3. In Section B.3 we plug the foregoing factoring result into Sudan's [Sud97] list-decoder for the RS, to show that it can be implemented in logspace-uniform $\mathcal{NC}$; and deduce a local list-decoder for the RM code in logspac-euniform $\mathcal{NC}$, relying on the reduction by Sudan, Trevisan and Vadhan [STV01] of this problem to the problem of list-decoding the RS code.

4. Finally, in Section B.4 we use the foregoing list-decoder for the Reed-Muller code with an additional simple argument to obtain a sample-aided worst-case to rare-case reduction for low-degree polynomials that is computable in logspace-uniform $\mathcal{NC}$.

Throughout the appendix, we assume that a bivariate polynomial $Q \in \mathbb{F}[x, y]$ with degree D is given as input by listing the coefficients of all of its $\binom{D+2}{2}$ monomials in the lexicographical order.[43] Similarly, a univariate polynomial $f \in \mathbb{F}[x]$ is also given as a list of coefficients, from the lowest power to the highest power.

B.1 Arithmetic and linear algebra in logspace-uniform $\mathcal{NC}$

Logspace-uniform $\mathcal{NC}$ circuits (and even more restricted circuit classes such as logtime-uniform $\mathcal{TC}^0$) can perform basic arithmetic operations, and solve standard linear-algebraic problems. We now recall several of these results that we will use in our proofs:

Lemma B.1 (arithmetic in logspace-uniform $\mathcal{NC}$)**.** *There exist logspace-uniform $\mathcal{NC}$ circuit families for the following problems over a prime field $\mathbb{F}$:*

[43]In more detail, the monomials are ordered first by their x-degrees, and then by their y-degrees.

- **Iterated addition.** *The input is a list of n field elements and the output is their sum. In this case the circuit is of linear size and of depth* $\log(n) \cdot \log(|\mathbb{F}|)$.

- **Iterated multiplication.** *The input is a list of n field elements and the output is their multiplication. In this case the circuit is of depth* $\log(n \cdot p)$.

- **Iterated addition of polynomials.** *The input is a list of univariate or bivariate polynomials over* $\mathbb{F}$ *with degree bound D, and the output is the polynomial that computes their sum.*

- **Multiplication of polynomials.** *The input is two univariate or bivariate polynomials over* $\mathbb{F}$ *with degree bound D, and the output is the polynomial that computes their multiplication.*

Proof. The circuit for iterated addition is just a tree that iteratively adds pairs of integers. Given n integers that are each represented by $\log(p)$ bits, the tree consists of $\log(n)$ levels, and in each level $i = 1, ..., \log(n)$ we add $n/2^{i+1}$ pairs of elements in size $(n/2^{i+1}) \cdot (\log(p))$ and depth $\log(p)$. The overall size is thus $O(n \cdot \log(p))$ and the overall depth is $\log(n) \cdot \log(p)$.

For iterated multiplication, Hesse, Allender and Barrington [HAB02] showed a construction in logtime-uniform $\mathcal{TC}^0$. Their construction is stronger, since any logtime-uniform $\mathcal{TC}^0$ circuit yields a logspace-uniform $\mathcal{NC}^1$ circuit, by replacing each linear threshold gate by a tree for addition and a top "greater-than" gadget.

Addition of two polynomials over $\mathbb{F}$ that are given as lists of coefficients reduces in logspace-uniform $\mathcal{NC}$ to addition of pairs of elements over $\mathbb{F}$. Additional of n polynomials over $\mathbb{F}$ reduces to addition of two polynomials over $\mathbb{F}$, via an addition tree as above (note that the degree bound remains identical throughout the procedure).

For multiplication of degree-D univariates we can again use the logtime-uniform $\mathcal{TC}^0$ circuits of [HAB02]. In case the polynomials are bivariate, we construct a circuit that maps the variable y to x^{2D}, multiplies the resulting two univariates, and transform them back to bivariates. ■

Systems of linear equations over a prime field $\mathbb{F}$ can be solved by logspace-uniform randomized $\mathcal{NC}$ circuits. As a first ingredient we verify that the determinant of a matrix can be computed in logspace-uniform $\mathcal{NC}$; then we show that systems with unique solutions can be solved in logspace-uniform $\mathcal{NC}$; and finally we reduce the problem of solving general linear systems to the problem of solving systems with unique solutions. The reduction was shown by Borodin, von zur Gathen, and Hopcroft [BGH82], while the unique solution case was established by Csanky [Csa76].

Lemma B.2 (computing the determinant in logspace-uniform $\mathcal{NC}$)**.** *There exists a logspace-uniform family of $\mathcal{NC}$ circuits that gets as input a matrix and computes its determinant.*

Proof. As shown in [Csa76], the determinant of a matrix can be computed in $\mathcal{NC}$. Our goal here is to further verify tat the $\mathcal{NC}$ algorithm for determinant in [Csa76] can indeed be implemented by log-space uniform $\mathcal{NC}$. To do so we closely follow the presentation in [Koz92, Lecture 31].

First, relying on Lemma B.1, matrix multiplication is in logspace-uniform $\mathcal{NC}$; and using repeated squaring, matrix powering to a polynomial power can also be computed in logspace-uniform $\mathcal{NC}$. We begin by establishing a simple logspace-uniform $\mathcal{NC}$ circuit for computing the inverse of an invertible lower triangular matrix A (that is, $A_{i,j} = 0$ for $i < j$). Without loss of generality assume A is of size $2^\ell \times 2^\ell$ for $\ell \in \mathbb{N}$, we write $A = \begin{bmatrix} B & 0 \\ C & D \end{bmatrix}$, where B, C and D are matrices of size $2^{\ell-1} \times 2^{\ell-1}$ and B and D are lower triangular. One can verify that $A^{-1} = \begin{bmatrix} B^{-1} & 0 \\ -D^{-1}CB^{-1} & D^{-1} \end{bmatrix}$. Hence, one can first compute B^{-1} and D^{-1} by recursion, and then compute A^{-1} by the aforementioned formula. This recursive algorithm can be straightforwardly implemented in logspace-uniform $\mathcal{NC}$, as it has ℓ levels (recall that A is of size $2^\ell \times 2^\ell$ and ℓ is logarithmic in terms of input size) and each level can be implemented by logspace-uniform $\mathcal{NC}$.

Next, given a general matrix A of size $n \times n$, the algorithm defines a vector $s = (s_1, s_2, \ldots, s_n) \in \mathbb{F}^n$ as follows: Denoting $s_0 = 1$, for every $k \in [n]$ we define[44]

$$s_k = \frac{1}{k} \cdot \left(\sum_{j=1}^{k} s_{k-j} \cdot \mathrm{tr}(A^j) \cdot (-1)^{j-1} \right)$$

In other words, plugging $s_0 = 1$, for each $k \in [n]$, we have the following linear equation

$$s_k - \frac{1}{k} \cdot \left(\sum_{j=1}^{k-1} s_{k-j} \cdot \mathrm{tr}(A^j) \cdot (-1)^{j-1} \right) = \mathrm{tr}(A^k) \cdot (-1)^{k-1}.$$

As proved in [Csa76], we have that $s_n = \det(A)$. Thus, our goal is to output s_n. To do so, note that there is a invertible lower triangular matrix $M \in \mathbb{F}^{n \times n}$ and a vector $c \in \mathbb{F}^n$ such that $Ms = c$; in more detail, we define

$$M_{k,k} = 1 \;\text{ and }\; M_{k,k-j} = \frac{1}{k} \cdot (-1)^j \cdot \mathrm{tr}(A^j) \text{ for } j \in [k-1]$$
$$c_k = \mathrm{tr}(A^k) \cdot (-1)^{k-1}\,.$$

By the above discussion both M and c can be computed by logspace-uniform $\mathcal{NC}$ circuits (since $|\mathbb{F}| > n$ and $1/k$ is defined as $k \neq 0$ for every $k \in [n]$). Also, we can solve the linear system $Ms = c$ by computing $s = M^{-1}c$, using the established logspace-uniform $\mathcal{NC}$ circuit for inverting lower triangular matrices. We output s_n. ■

Lemma B.3 (solving linear systems with unique solution in logspace-uniform $\mathcal{NC}$)**.** *There exists a logspace-uniform family of $\mathcal{NC}$ circuits that gets as input a linear system with n variables and n equations over a prime field $\mathbb{F}$ such that $|\mathbb{F}| > n$ and the linear system is guaranteed to have a unique solution, and outputs the unique solution.*

Proof. Denoting the linear system by $Ax = b$, and recalling that it has a unique solution (i.e., that A has full rank), we know that the solution is $x = A^{-1}b$. By Cramer's rule, matrix inversion reduces to computing the determinants of its minors (as well as the determinant of the matrix itself), and this reduction can indeed be computed in logspace-uniform $\mathcal{NC}$ (since each output element is the determinant of a minor of a predetermined submatrix, divided by $\pm$ of the determinant of the matrix itself). The claim follows by Lemma B.2. ■

[44] Recall that for an $n \times n$ matrix M, $\mathrm{tr}(M)$ is defined as $\sum_{i=1}^{n} M_{i,i}$.

Lemma B.4 (solving general linear systems in logspace-uniform randomized $\mathcal{NC}$)**.** *There exists a logspace-uniform family of randomized $\mathcal{NC}$ circuits that get as input a linear system over a prime field $\mathbb{F}$, and with probability at least $1 - 2^{-|\mathbb{F}|}$ output a solution if one exists.*

Proof. The proof of [BGH82, Theorem 5] reduces solving general linear systems $Ax = b$ where $A = n \times n$ to solving systems with unique solutions. Their reduction works as follows:

1. Find a basis for the column-space of A among the columns, and find a basis for the row-space of A among the rows.
2. Solve the linear system corresponding to the submatrix of bases, which has a unique solution $v \in \mathbb{F}^{\mathtt{rank}(A)}$.
3. Extend v to a vector $y \in \mathbb{F}^n$ by placing zeroes in the yet-unspecified $n - \mathtt{rank}(A)$ locations. If $Ay = b$, output y, otherwise output $\perp$.

By Lemmas B.3 and B.1 we can perform the second and third steps (respectively) in logspace-uniform $\mathcal{NC}$. It remains to verify that the first step is in logspace-uniform probabilistic $\mathcal{NC}$.

To find a basis among a set of vectors $\{v_1, ..., v_n\}$ for their span, it suffices to include v_i iff $\mathtt{rank}(\{v_1, ..., v_{i-1}\}) < \mathtt{rank}(\{v_1, ..., v_i\})$ (observe that this reduction from finding a basis to computing the rank is in logspace-uniform $\mathcal{NC}$). Following the proof of [BGH82, Theorem 3], we can probabilistically compute the rank of an $m \times m$ matrix A' by taking the maximum among the outputs of $O(1)$ parallel repetitions of the following experiment:

1. Choose two random $m \times m$ matrices B, C.
2. For all $i \in [m]$, compute the determinant of the principal $i \times i$ minor of $BA'C$.
3. Output $\max\{i \in [n] : f_i \neq 0\}$, or 0 if no such i exists.

Relying on Lemma B.2, the entire procedure above can be carried out by logspace-uniform probabilistic $\mathcal{NC}$ circuits. ■

B.2 Factoring linear factors from bivariates

Our goal in this section is to construct a logspace-uniform $\mathcal{NC}$ circuit that gets as input a bivariate polynomial $Q \in \mathbb{F}[x, y]$ and finds all of its factors that are of the form $y - p(x)$ for some univariate polynomial $p(x)$. Specifically, we prove the following:

Theorem B.5 (factoring linear factors from bivariate polynomials in logspace-uniform $\mathcal{NC}$)**.** *Let $\mathbb{F}$ be a prime field and let $D \geq 1$ be an integer such that $|\mathbb{F}| > 2D^2$. Then, there exists an algorithm $\mathsf{factorize}_{D,\mathbb{F}}$ that gets as input $Q \in \mathbb{F}[x, y]$ with degree at most D and advice $(r, a, t) \in [D] \times \mathbb{F}^2$ and satisfies the following:*

1. *For every $\tau(x) \in \mathbb{F}[x]$ such that $(y - \tau(x))$ is a factor of Q, there exists $(r, a, t) \in [D] \times \mathbb{F}^2$ such that $\mathsf{factorize}_{D,\mathbb{F}}(Q, r, a, t) = \tau(x)$.*
2. *The algorithm $\mathsf{factorize}_{D,\mathbb{F}}$ can be implemented by logspace-uniform $\mathcal{NC}$ circuits.*

High-level description of the algorithm

The idea behind the above theorem is to use the Hensel lifting technique, adapted to our particular setting. Loosely speaking, given a bivariate polynomial $Q(x,y)$ and a factor $(y-t)$ of $Q(0,y) \in \mathbb{F}[y]$, one can "lift" $(y-t)$ to a factor $(y-\tau(x))$ of $Q(x,y) \in \mathbb{F}[x,y]$, under the condition that $y-t$ has multiplicity exactly 1 in $Q(0,y)$, and $t=\tau(0)$ (see Proposition B.11). While we do not know $t=\tau(0)$ (as we do not know τ in advance), our circuit simply tries all possible $t \in \mathbb{F}$ in parallel, as we are allowed to use size $\mathrm{poly}(|\mathbb{F}|)$.

To deal with the requirement that $y-t$ has multiplicity 1 in $Q(0,y)$, we atke the following preprocessing steps. Suppose that Q has a factor $(y-\tau(x))$ with multiplicity r. We first observe that $y-\tau(x)$ would be a factor of $\frac{\partial^{r-1}Q}{\partial^{r-1}y}$ with multiplicity 1 (see Lemma B.8). So we enumerate all possible multiplicities r at the beginning, and deal with all $Q^{(r-1)} = \frac{\partial^{r-1}Q}{\partial^{r-1}y}$ (in parallel) instead of Q. But this does not guarantee that $y-\tau(0)$ has multiplicity 1 in $Q(0,y)$.[45] We further consider all "shifted versions" of $Q^{(r-1)}$, defined by $Q_a^{(r-1)} = Q^{(r-1)}(x+a,y)$. Note that factor $y-\tau(x)$ of $Q^{(r-1)}$ is in one-to-one correspondence to factor $y-\tau(x-a)$ of $Q_a^{(r-1)}$. We show that there must exist a shift a such that $y-\tau(x-a)$ has multiplicity 1 in $Q_a^{(r-1)}(0,y)$ (see Lemma B.9). Hence, to ensure the multiplicity condition, we enumerate all possible powers r and all possible shifts a, and then apply the Hensel lifting to every derived polynomials $Q_a^{(r-1)}$ (see Algorithm 3).

B.2.1 Preliminaries

In this section, when we refer to rings we always mean commutative rings. For a polynomial $Q \in \mathbb{F}[x,y]$ such that $Q=\sum_{a,b} c_{a,b}x^a y^b$, we use $\deg(Q)$, $\deg_x(Q)$ and $\deg_y(Q)$ to denote the degree of Q, the x-degree of Q and the y-degree of Q, respectively; that is, $\deg(Q)$ is defined as $\max\{a+b \mid c_{a,b} \neq 0\}$, and $\deg_x(Q)$ (resp. $\deg_y(Q)$) is defined as $\max\{a \mid c_{a,b} \neq 0\}$ (resp. $\max\{b \mid c_{a,b} \neq 0\}$).

For a ring R and an ideal $I \subseteq R$, we say that elements $g,h \in R$ are coprime mod I if there exist $a,b \in R$ such that $ag+bh \equiv 1 \pmod{I}$.

Properties of polynomials

Towards presenting the algorithm, we state several elementary facts about polynomials.

Lemma B.6 (division by linear factors)**.** *Let R be a unique factorization domain, let $f \in R[y]$, and let $t \in R$. Then, there exists a unique $h \in R[y]$ such that*

$$f(y)-f(t)=(y-t)h(y).$$

We also need the following algorithmic version of a special case of Lemma B.6, in which $R=\mathbb{F}[x]$.

Lemma B.7 (algorithmic division by linear factors)**.** *Let $\mathbb{F}$ be a prime field, and let $D \geq 1$ be an integer. There is an algorithm $\mathsf{SimpleDivision}_{D,\mathbb{F}}$ that gets as input $f \in \mathbb{F}[x,y]$ with degree at most D and*

[45] For example, it could be the case that Q has another factor $y-\tilde{\tau}(x)$ such that $\tau(x) \neq \tilde{\tau}(x)$ but $\tilde{\tau}(0)=\tau(0)$. In this case $y-\tau(0)$ would have multiplicity at least 2 in $Q(0,y)$.

$\tau \in \mathbb{F}[x]$, *and outputs* $h \in \mathbb{F}[x,y]$ *such that*

$$f(x,y) = (y - \tau(x)) \cdot h(x,y) + f(x, \tau(x)).$$

Moreover, $\mathsf{SimpleDivision}_{D,\mathbb{F}}$ *can be implemented by logspace-uniform* $\mathcal{NC}$ *circuits.*

Proof. First, by Lemma B.6 (with the same f and $R = \mathbb{F}[x]$ and $t = \tau(x)$), we know that there exists a unique h satisfying our requirement. We get as input

$$f(x,y) = \sum_{a,b\colon a+b \le D} c_{a,b} x^a y^b.$$

Letting $\gamma(x) = f(x,\tau(x)) = \sum_{a,b\colon a+b\le D} c_{a,b} x^a \tau(x)^b$, note that we can compute a representation of γ as a list of monomials by logspace-uniform $\mathcal{NC}$ circuits. In more detail, we can use polynomial powering to compute $c_{a,b}x^a\tau(x)^b$ (multiplying by $c_{a,b}x^a$ is simple), and then sum up all the resulting $O(D^2)$ polynomials by polynomial addition, using Lemma B.1.

Now we have

$$f(x,y) - \gamma(x) = (y - \tau(x)) \cdot h(x,y). \tag{1}$$

Note that we have $\deg(h) \le \deg(f) - 1 \le D - 1$. Examining the requirement (1), since we already know f, γ and τ, we can treat their coefficients as known constants, and (1) can be equivalently written as $\binom{D+1}{2}$ linear equations, involving coefficients of h as the only variables.

In more detail, denoting $h = \sum_{a+b\le D-1} h_{a,b} x^a y^b$, the equation implies that

$$f(x,y) - \gamma(x) = (y - \tau(x)) \cdot \sum_{a+b \le D-1} h_{a,b} x^a y^b = \sum h_{a,b} x^a y^{b+1} - \sum h_{a,b} x^a \tau(x) y^b.$$

After gathering coefficients on the RHS, we are left with a pair of equal polynomials (one on the RHS and one on the LHS), and each coefficient of the polynomial on the RHS is a linear combination of the $h_{a,b}$. Since we already computed representations of f, of γ, and of τ, we treat their coefficients as known constants, and this yields a linear system with the $h_{a,b}$'s as variables and with $\binom{D+1}{2}$ equations.

By the above discussions, (1) has a unique solution, and therefore the linear system we just constructed also has a unique solution. We can construct the system in logspace-uniform $\mathcal{NC}$ by adding field elements (using Lemma B.1), and solve the system in logspace-uniform $\mathcal{NC}$ (using Lemma B.3). ■

Lemma B.8. *Let* $Q \in \mathbb{F}[x,y]$ *such that* $\deg(Q) < \mathsf{char}(\mathbb{F})$, *and let* $\tau \in \mathbb{F}[x]$ *such that* $y - \tau(x)$ *is a factor of* Q *with multiplicity* $r \ge 1$. *Then,* $y - \tau(x)$ *is a factor of* $\frac{\partial^{r-1} Q}{\partial^{r-1} y}$ *with multiplicity* 1.

Proof. Let us prove the lemma by induction. Clearly it holds when $r = 1$. For $r \ge 2$, suppose the lemma holds when $r - 1$. Since $y - \tau(x)$ is a factor of Q with multiplicity r, we can write

$$Q(x,y) = (y - \tau(x))^r \cdot H(x,y)$$

such that $H \in \mathbb{F}[x,y]$, and $y-\tau(x)$ does not divide H. By the chain rule of partial derivatives,

$$\frac{\partial Q}{\partial y}(x,y) = \left[(y-\tau(x))^r \cdot \frac{\partial H}{\partial y}(x,y)\right] + \left[r \cdot (y-\tau(x))^{r-1} \cdot H(x,y)\right]. \tag{2}$$

Now, observe that $r \neq 0$ (since $\mathsf{char}(\mathbb{F}) > \deg(Q) \geq r$). This means that $(y-\tau(x))^r$ does not divide (2), and consequently $y-\tau(x)$ is a factor of $\frac{\partial Q}{\partial y}$ with multiplicity $r-1$. The lemma then follows from the induction hypothesis. ■

Lemma B.9. *Let $Q \in \mathbb{F}[x,y]$ such that $2\deg(Q)^2 < |\mathbb{F}|$, and let $(y-\tau(x))$ be a factor of Q with multiplicity 1. Then, there exists $a \in \mathbb{F}$ such that $(y-\tau(a))$ is a factor of $Q(a,y) \in \mathbb{F}[y]$ with multiplicity 1.*

Proof. Let $H \in \mathbb{F}[x,y]$ such that

$$Q(x,y) = (y-\tau(x)) \cdot H(x,y).$$

Since $(y-\tau(x))$ has multiplicity 1 in $Q(x,y)$, it means that $(y-\tau(x))$ does not divide $H(x,y)$. By Lemma B.6 with $f = H(x,y) \in (\mathbb{F}[x])[y]$ and $t = \tau(x)$, we have

$$H(x,y) = \tilde{H}(x,y) \cdot (y-\tau(x)) + \gamma(x),$$

for $\gamma(x) = H(x,\tau(x))$. Note that $\gamma(x)$ has degree at most $\deg_x(H) + \deg_y(H) \cdot \deg(\tau) \leq \deg(Q) + \deg(Q)^2 \leq 2\deg(Q)^2$. Also note that $\gamma(x)$ is not a zero polynomial.

For any $a \in \mathbb{F}$, plugging in $x = a$, we have

$$Q(a,y) = (y-\tau(a)) \cdot H(a,y) = (y-\tau(a)) \cdot (\tilde{H}(a,y) \cdot (y-\tau(a)) + \gamma(a)).$$

As long as a satisfies $\gamma(a) \neq 0$, we have that $(y-\tau(a))$ is a factor of $Q(a,y)$ with multiplicity 1. (Also note that $\tilde{H}(a,y) \cdot (y-\tau(a)) + \gamma(a)$ is a non-zero polynomial by examining the highest y-power.) Such an a exists as $\deg(\gamma) \leq 2\deg(Q)^2 < |\mathbb{F}|$. ■

Hensel lifting

The following lemma capturing the well-known Hensel lifting technique will be crucial for our algorithms.

Lemma B.10. *Let R be a ring, let $I \subseteq R$ be an ideal, and let $f \in R$. Suppose that there are elements $g, h \in R$ such that*

(H1) $f \equiv gh \pmod{I}$,

(H2) *g and h are coprime mod I.*

Then, the following holds:

1. ***Lifting:*** *There exist $\tilde{g}, \tilde{h} \in R$ such that*

   (C1) $f \equiv \tilde{g}\tilde{h} \pmod{I^2}$,

(C2) $\tilde{g}$ *and* $\tilde{h}$ *are coprime mod* I^2.

(C3) $g \equiv \tilde{g} \pmod{I}$ *and* $h \equiv \tilde{h} \pmod{I}$

2. ***Uniqueness:*** *For every elements* $\tilde{g}, \tilde{h} \in R$ *such that (C1) – (C3) holds, if there are other two elements* $g', h' \in R$ *satisfying (C1) and (C3), then there exists* $u \in I$ *such that*

$$g' \equiv \tilde{g}(1+u) \pmod{I^2}, \quad \textit{and} \quad h' \equiv \tilde{h}(1-u) \pmod{I^2}.$$

For completeness, we give below a standard proof of this lemma, following [Sap17, Theorem 12.1] and [Sud15, Lemma 4].

Proof. Denote $f = gh + q$ for some $q \in I$, and let $a, b \in R$ and $r \in I$ such that $ag + bh = 1 + r$. We define

$$\tilde{g} = g + bq, \quad \tilde{h} = h + aq \quad .$$

Observe that $\tilde{g} = g \pmod{I}$ and $\tilde{h} = h \pmod{I}$, and also that

$$\begin{aligned}\tilde{g}\tilde{h} &= gh + gaq + hbq + baq^2 \\ &= f - q + q(1+r) + baq^2 \\ &= f + qr + baq^2 \\ &= f \pmod{I^2}.\end{aligned}$$

To show that suitable $\tilde{a}, \tilde{b}$ exist, note that for some $s \in I$ it holds that $a\tilde{g} + b\tilde{h} = ag + bh + abq + baq = 1 + r + 2qab = 1 + s$; we define $\tilde{a} = a(1-s)$ and $\tilde{b} = b(1-s)$, and note that $\tilde{a}\tilde{g} + \tilde{b}\tilde{h} = (1-s)(a\tilde{g} + b\tilde{h}) = 1 - s^2 = 1 \pmod{I^2}$.

For the uniqueness part, we will now assume (g_1, h_1, a_1, b_2) satisfy (C1) - (C3), and let (g_2, h_2) be two elements satisfying (C1) and (C3). Since both (g_1, h_1) and (g_2, h_2) satisfy (C3), there exists $\alpha, \beta \in I$ such that $g_2 = g_1 + \alpha$ and $h_2 = h_1 + \beta$. Also, from (C1), we have that $g_1h_1 \equiv f \equiv g_2h_2 \pmod{I^2}$. Then, it follows that

$$\begin{aligned}0 \equiv g_2h_2 - g_1h_1 &\equiv (g_1 + \alpha)(h_1 + \beta) - g_1h_1 \pmod{I^2} \\ &\equiv \alpha h_1 + g_1\beta \pmod{I^2} \qquad (\alpha\beta \in I^2)\end{aligned}$$

which implies that

$$\begin{aligned}&b_1\alpha h_1 + b_1g_1\beta \equiv 0 \pmod{I^2} \\ \implies &\alpha(1 - a_1g_1) + b_1g_1\beta \equiv 0 \pmod{I^2} \qquad \text{(see below)} \\ \implies &\alpha = g_1(a_1\alpha - b_1\beta) \pmod{I^2}\end{aligned}$$

where the equality $b_1h_1 \equiv 1 - a_1g_1 \pmod{I^2}$ above is since $a_1g_1 + b_1h_1 \equiv 1 \pmod{I^2}$.

Let $u = a_1\alpha - b_1\beta$, note that since $\alpha, \beta \in I$, we also have that $u \in I$. The above shows that $\alpha = g_1u \pmod{I^2}$, and since $g_2 = g_1 + \alpha$, it follows that

$$g_2 = g_1(1+u) \pmod{I^2}.$$

By a symmetric argument we have that $\beta = h_1(b_1\beta - a_1\alpha) = -h_1 u \pmod{I^2}$, and since $h_2 = h_1 + \beta$ we have that $h_2 = h_1(1-u)$. ■

The foregoing Hensel lifting procedure can be concisely described by the following algorithm.

Algorithm 1 $\mathsf{HenselLift}_{R,I}(f, g, h, a, b)$

Parameters: A ring R and an ideal I.

Input: Given $f, g, h \in R$ satisfying (H1) and (H2), and $a, b \in R$ such that $ag + bh \equiv 1 \pmod{I}$.

```
1: $q \leftarrow f - gh$
2: $\tilde{g} \leftarrow g + bq, \quad \tilde{h} = h + aq$
3: $s \leftarrow a\tilde{g} + b\tilde{h} - 1$
4: $\tilde{a} \leftarrow a(1-s), \quad \tilde{b} = b(1-s)$
```

Output: A tuple $(\tilde{g}, \tilde{h}, \tilde{a}, \tilde{b})$ satisfying (C1) - (C3) and $\tilde{a}\tilde{g} + \tilde{b}\tilde{h} \equiv 1 \pmod{I^2}$.

B.2.2 Hensel lifting for factoring linear terms from polynomials

We now describe a modification of the generic Hensel lifting procedure that is particularly suited to our setting. Specifically, in our setting the ring R is $\mathbb{F}[x, y]$, and we are looking to factor a given polynomial $Q(x, y)$ such that one factor will be of the form $(y - \tau(x))$ (rather than to factor the polynomial arbitrarily). Moreover, we wish to perform this procedure efficiently, by logspace-uniform circuits of low depth.

Proposition B.11. *Let $\mathbb{F}$ be a prime field and let $D \geq 1$ be an integer. Then, there exists a procedure* $\mathsf{PolyLift}_{D,\mathbb{F}}$ *that gets as input $Q \in \mathbb{F}[x, y]$ of degree at most D and $t \in \mathbb{F}$, and satisfies the following.*

1. *If Q has a factor $y - \tau(x)$ such that $t = \tau(0)$, and the multiplicity of $y - t$ in $Q(0, y)$ is 1. Then,* $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$ *outputs $\tau(x)$.*

2. *The procedure* $\mathsf{PolyLift}$ *can be implemented by logspace-uniform NC circuits.*

We stress that the assumption in Proposition B.11 that the multiplicity of $y - t$ in $Q(0, y)$ is 1 implies that there is only one factor $y - \tau(x)$ satisfying $t = \tau(0)$. The first item in Proposition B.11 asserts that $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$ outputs $\tau(x)$ (i.e., essentially outputs this factor). In the rest of the section we prove Proposition B.11, by first describing $\mathsf{PolyLift}_{D,\mathbb{F}}$ and then analyzing it.

Description of $\mathsf{PolyLift}_{D,\mathbb{F}}$

Condition check. $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$ first checks whether $(y - t)$ is a factor of $Q(0, y)$ with multiplicity 1. It returns $\perp$ immediately if the condition fails to hold. Otherwise, since $(y - t)$ is a factor of $Q(0, y)$, there is a unique polynomial $h(y) \in \mathbb{F}[y]$ such that

$$Q(0, y) = (y - t) \cdot h(y).$$

High-level description of the main loop. Let K be the smallest integer so that $2^K > D \geq \deg_x(Q)$. The goal of the algorithm $\mathsf{PolyLift}_{D,\mathbb{F}}$ is to iteratively construct, for every $0 \leq k \leq K$, a polynomial $p_k \in \mathbb{F}[x]$ and $q_k, a_k, b_k \in \mathbb{F}[x,y]$ such that

$$Q(x,y) \equiv (y - p_k(x)) \cdot q_k(x,y) \pmod{x^{2^k}}, \tag{3}$$

$$a_k(y - p_k(x)) + b_k q_k(x,y) \equiv 1 \pmod{x^{2^k}}. \tag{4}$$

$$y - p_k(x) \equiv y - t \pmod{x} \quad \text{and} \quad q_k(x,y) \equiv h(y) \pmod{x}. \tag{5}$$

We will also show that, if Q has a factor of the form $(y - \tau(x))$ such that $\tau(0) = t$, then for every k it holds that $\tau(x) \equiv p_k(x) \pmod{x^{2^k}}$. To put it succinctly, we will show that

$$\tau(x) \equiv p_k(x) \pmod{x^{2^k}} \quad \text{if } y - \tau(x) \text{ divides } Q \text{ and } \tau(0) = t. \tag{6}$$

The algorithm will finally return $p_K(x)$.

Base step. The goal of the base case is to construct $p_0 \in \mathbb{F}[x]$, $q_0, a_0, b_0 \in \mathbb{F}[x,y]$ so that they satisfy (3), (4), (5) and (6) for $k = 0$.

Letting $q_0(x,y) = h(y)$ and $p_0(x) = t$ (they clearly satisfy (5) for $k = 0$), we have

$$Q(x,y) \equiv (y - p_0(x)) \cdot q_0(x,y) \pmod{x},$$

which establishes (3) for $k = 0$.

Next we show how to construct two polynomials $a_0, b_0 \in \mathbb{F}[y]$ so that

$$a_0(y - t) + b_0 h(y) = 1,$$

thereby establishing (4) for $k = 0$. By Lemma B.6, we can write $h(y) = (y - t) \cdot \tilde{h}(y) + \beta$, where $\beta = h(t) \neq 0$. (We know that $(y - t)$ does not divide $h(y)$, since $(y - t)$ has multiplicity 1 in $Q(0,y)$ and $Q(0,y) = (y - t)h(y)$.) Then we can simply set $a_0 = (-\beta^{-1}) \cdot \tilde{h}(y)$ and $b_0 = \beta^{-1}$.

Finally, suppose that Q has a factor $(y - \tau(x))$ such that $\tau(0) = t$. Then clearly $p_0(x) \equiv t \equiv \tau(x) \pmod{x}$, establishing (6) for $k = 0$.

Induction. For an integer $1 \leq k \leq K$, suppose that we have $p_{k-1} \in \mathbb{F}[x]$, $q_{k-1}, a_{k-1}, b_{k-1} \in \mathbb{F}[x,y]$ such that (3), (4), (5) and (6) hold for $k - 1$. In the following we show how to construct (p_k, q_k, a_k, b_k) so that (3), (4), (5) and (6) hold for k. We do so as follows:

1. Construction of $\tilde{g}, \tilde{h}, \tilde{a}, \tilde{b}$. For this part we will simply apply the generic Hensel lifing algorithm $\mathsf{HenselLift}$. More precisely, let $R = \mathbb{F}[x,y]$ and $I = (x^{2^{k-1}})$, we let

$$(\tilde{g}, \tilde{h}, \tilde{a}, \tilde{b}) = \mathsf{HenselLift}_{R,I}(Q, y - p_{k-1}, q_{k-1}, a_{k-1}, b_{k-1}).$$

By Lemma B.10, $\tilde{g}, \tilde{h}, \tilde{a}, \tilde{b}$ have the following properties:

$$Q \equiv \tilde{g}\tilde{h} \pmod{x^{2^k}}, \quad \tilde{g} \equiv y - t \pmod{x}, \quad \text{and} \quad \tilde{h} \equiv h(y) \pmod{x},$$

and

$$\tilde{a}\tilde{g} + \tilde{b}\tilde{h} \equiv 1 \pmod{x^{2^k}}.$$

(Note that Lemma B.10 indeed shows that $\tilde{g} \equiv y - p_{k-1}(x) \pmod{x^{2^{k-1}}}$, which implies $\tilde{g} \equiv y - t \pmod{x}$ together with our induction hypothesis (5). Similarly we also have $\tilde{h} \equiv h(y) \pmod{x}$ from $\tilde{h} \equiv q_{k-1}(x) \pmod{x^{2^{k-1}}}$ and our induction hypothesis (5).)

Additionally, following the algorithm of $\mathsf{HenselLift}_{R,I}(Q, y - p_{k-1}, q_{k-1}, a_{k-1}, b_{k-1})$, we also set

$$q = Q - (y - p_{k-1})q_{k-1}. \tag{7}$$

and we have that

$$\tilde{g} = (y - p_{k-1}) + b_{k-1}q \text{ and } \tilde{h} = q_{k-1} + a_{k-1}q, \tag{8}$$

according to $\mathsf{HenselLift}_{R,I}(Q, y - p_{k-1}, q_{k-1}, a_{k-1}, b_{k-1})$.

2. Construction of p_k, q_k, a_k, b_k. Next, we show how to modify $\tilde{g}$ and $\tilde{h}$ to construct p_k and q_k. For a suitable $r \in \mathbb{F}[x, y]$ such that $x^{2^{k-1}}|r$ that will be defined in a moment, we will show that there exists p_k such that

$$\tilde{g}(x, y) \cdot (1 - r) \equiv y - p_k(x) \pmod{x^{2^k}}$$

and we will define $q_k = \tilde{h} \cdot (1 + r)$, which means that

$$\tilde{h}(x, y) \cdot (1 + r) \equiv q_k(x, y) \pmod{x^{2^k}}.$$

The key observation here is that for a suitable r, since $x^{2^{k-1}}|r$, we have that

$$(1 - r) \cdot (1 + r) = 1 - r^2 \equiv 1 \pmod{x^{2^k}},$$

and hence

$$(y - p_k(x)) \cdot q_k(x, y) \equiv \tilde{g}(x, y)\tilde{h}(x, y) \equiv 1 \pmod{x^{2^k}}.$$

We now proceed as follows:

- *Defining r.* Let $\tau(x, y) = (b_{k-1}q)/x^{2^{k-1}}$. (Recall that $x^{2^{k-1}}|q$ from (7)) We think of $\tau(x, y)$ as an element in $(\mathbb{F}[x])[y]$. Letting $R = \mathbb{F}[x]$, by Lemma B.7, we can compute $\bar{h}(x, y) \in \mathbb{F}[x, y]$ and $\gamma(x) = \tau(x, p_{k-1}(x))$ in logspace-uniform NC such that

$$\tau(x, y) = (y - p_{k-1}(x)) \cdot \bar{h}(x, y) + \gamma(x).$$

  Plugging in $b_{k-1}q$, we have

$$\begin{aligned}(b_{k-1}q)(x, y) &= [(y - p_{k-1}(x))\bar{h}(x, y) + \gamma(x)] \cdot x^{2^{k-1}}. \\ &= [g\bar{h}(x, y) + \gamma(x)] \cdot x^{2^{k-1}}\end{aligned}$$

  Now we set $r = x^{2^{k-1}} \cdot \bar{h}(x, y)$.

- *Verifying that $\tilde{g} \cdot (1 - r)$ is of the form $y - p_k$.* We now verify that

$$\tilde{g} \cdot (1 - r) \mod x^{2^k}$$

can indeed be written as a polynomial of the form $y - p_k(x)$, as follows:

$$\begin{aligned}
&\tilde{g}(1-r) \\
=&(g + b_{k-1}q)(1-r) && \text{(by (8))} \\
=&(g + [g\bar{h}(x,y) + \gamma(x)] \cdot x^{2^{k-1}})(1 - x^{2^{k-1}} \cdot \bar{h}(x,y)) \\
\equiv& g + [g\bar{h}(x,y) + \gamma(x)] \cdot x^{2^{k-1}} - x^{2^{k-1}} \cdot \bar{h}(x,y) \cdot g \pmod{x^{2^k}} \\
\equiv& g + \gamma(x) \cdot x^{2^{k-1}} \pmod{x^{2^k}} \\
\equiv& y - p_{k-1}(x) + \gamma(x) \cdot x^{2^{k-1}} \pmod{x^{2^k}}.
\end{aligned}$$

Hence, we can set

$$p_k(x) = p_{k-1}(x) - \gamma(x) \cdot x^{2^{k-1}} \mod x^{2^k}$$

- *Modifying $\tilde{a}, \tilde{b}$ to a_k, b_k accordingly.* Finally, we set $a_k = \tilde{a} \cdot (1+r) \mod x^{2^k}$ and $b_k = \tilde{b} \cdot (1-r) \mod x^{2^k}$, we have

$$\begin{aligned}
a_k(y - p_k(x)) + b_k q_k(x,y) &\equiv \tilde{a} \cdot (1+r)\tilde{g} \cdot (1-r) + \tilde{b} \cdot (1-r)\tilde{h} \cdot (1+r) \pmod{x^{2^k}} \\
&\equiv (1-r^2) \cdot (\tilde{a}\tilde{g} + \tilde{b}\tilde{h}) \pmod{x^{2^k}} \\
&\equiv 1 \pmod{x^{2^k}}.
\end{aligned}$$

3. Verifying (6). Suppose that Q has a factor $(y - \tau(x))$ such that $\tau(0) = t$, and denote $Q(x,y) = (y - \tau(x)) \cdot H(x,y)$ for some $H \in \mathbb{F}[x,y]$. Since (6) holds for $k-1$, we have that $y - \tau(x) \equiv y - p_{k-1}(x) \pmod{x^{2^{k-1}}}$.

We first show that $H(x,y) \equiv q_{k-1}(x,y) \pmod{x^{2^{k-1}}}$. To see this, note that our assumption $y - \tau(x) \equiv y - p_{k-1}(x) \pmod{x^{2^{k-1}}}$ implies that

$$\begin{aligned}
Q = (y - \tau(x)) \cdot H(x,y) &\equiv (y - p_{k-1}(x)) \cdot q_{k-1}(x,y) \pmod{x^{2^{k-1}}} \\
&\equiv (y - \tau(x)) \cdot q_{k-1}(x,y) \pmod{x^{2^{k-1}}},
\end{aligned}$$

which further implies that

$$(y - \tau(x)) \cdot (H(x,y) - q_{k-1}(x,y)) \equiv 0 \pmod{x^{2^{k-1}}}.$$

Examining the highest y-power at the left-hand, the above is only possible when $H(x,y) - q_{k-1}(x,y) \equiv 0 \pmod{x^{2^{k-1}}}$, which proves our claim.

Now, to prove that $\tau(x) \equiv p_k(x) \pmod{x^{2^k}}$ we instantiate Lemma B.10 with the following parameters:

$$\begin{aligned}
&R = \mathbb{F}[x,y], && I = (x^{2^{k-1}}), && f = Q, \\
&g = y - p_{k-1}, && h = q_{k-1}, \\
&\tilde{g} = y - p_k, && \tilde{h} = q_k, \\
&g' = y - \tau(x), && h' = H.
\end{aligned}$$

Note that $(\tilde{g}, \tilde{h})$ satisfy (C1) – (C3) of Lemma B.10 and that (g', h') satisfy (C1) and (C3) of Lemma B.10. It follows that there exists $u \in I$ (that is, $x^{2^{k-1}}$ divides u) such that

$$(y - \tau(x)) \equiv (y - p_k(x)) \cdot (1 + u) \pmod{x^{2^k}}. \tag{9}$$

Again, we will examine the highest y-power on both sides to show that $\tau(x) \equiv p_k(x) \pmod{x^{2^k}}$. First, one can see that $\deg_y(u) = 0$, as otherwise the right side of (9) would have y-degree greater than 1, but the left side of (9) has y-degree exactly 1. Next, looking at the coefficients of y at both sides, we have $1 \equiv 1 + u \pmod{x^{2^k}}$, meaning that $u \equiv 0 \pmod{x^{2^k}}$, and consequently $\tau(x) \equiv p_k(x) \pmod{x^{2^k}}$. This verifies the condition (6).

Output. Finally, $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$ returns $p_K(x)$. The whole algorithm can be succinctly descried as follows.

Algorithm 2 $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$

Parameters: D is the maximum degree parameter and $\mathbb{F}$ is a prime field.
Input: Given $Q \in \mathbb{F}[x, y]$ with degree at most D and $t \in \mathbb{F}$.

```
1: if $y - t$ does not divide $Q(0, y)$ or $(y - t)^2$ divides $Q(0, y)$ then return $\perp$    ▷ Condition check
2: end if
3: $h(y) \leftarrow Q(0, y)/(y - t)$    ▷ Base step
4: $q_0 \leftarrow h, \quad p_0 \leftarrow t$
5: $\tilde{h} \leftarrow (h(y) - h(t))/(y - t), \quad \beta \leftarrow h(t)$
6: $a_0 \leftarrow -\beta^{-1}\tilde{h}, \quad b_0 \leftarrow \beta^{-1}$
7: $K = \lceil \log(D + 1) \rceil$.    ▷ Induction step
8: for $k \leftarrow 1, \ldots, K$ do
9:     $(\tilde{g}, \tilde{h}, \tilde{a}, \tilde{b}) \leftarrow \mathsf{HenselLift}_{\mathbb{F}[x,y],(x^{2^{k-1}})}(Q, y - p_{k-1}, q_{k-1}, a_{k-1}, b_{k-1})$.    ▷ Computing $\tilde{g}, \tilde{h}, \tilde{a}, \tilde{b}$
10:    $q \leftarrow Q - (y - p_{k-1})q_{k-1}$
11:    $\tau \leftarrow b_{k-1}q/x^{2^{k-1}}$    ▷ Start computing $p_k, q_k, a_k, b_k$
12:    $\gamma(x) \leftarrow \tau(x, p_{k-1}(x)), \quad \bar{h} \leftarrow (\tau - \gamma)/(y - p_{k-1})$
13:    $r \leftarrow x^{2^{k-1}}\bar{h}$
14:    $p_k \leftarrow p_{k-1} - \gamma \cdot x^{2^{k-1}} \mod x^{2^k}, \quad q_k \leftarrow \tilde{h} \cdot (1 + r) \mod x^{2^k}$
15:    $a_k \leftarrow \tilde{a} \cdot (1 + r) \mod x^{2^k}, \quad b_k \leftarrow \tilde{b} \cdot (1 - r) \mod x^{2^k}$
16: end for
17: return $p_K$
```

Analysis of $\mathsf{PolyLift}_{D,\mathbb{F}}$

We now analyze the algorithm $\mathsf{PolyLift}_{D,\mathbb{F}}$ and prove Proposition B.11.

Proof of Proposition B.11. We first prove the correctness of the algorithm (i.e., the first item in the statement) and then argue about its efficiency (i.e., prove the second item in the statement).

Correctness. Let $t = \tau(0)$. Since the multiplicity of $y - t$ in $Q(0, y)$ is 1, the condition check phase of $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$ passes successfully. Next, recall that $\mathsf{PolyLift}_{D,\mathbb{F}}(Q, t)$ sets K so that $2^K >$

$\deg(Q)$. By (6) and the assumption, we have that $\tau(x) \equiv p_K(x) \pmod{x^{2^K}}$. Since $2^K > \deg(\tau)$ as well, this means that $\tau(x) = p_K(x)$.

Complexity. According to Algorithm 2 (also with its subroutine Algorithm 1), $\mathsf{PolyLift}_{D,\mathbb{F}}(Q,t)$ proceeds in $O(\log D)$ iterations, and each iteration requires a constant number of arithmetic operations on polynomials from $\mathbb{F}[x,y]$ with at most $O(D)$ degree. Relying on Lemma B.1, multiplication, addition, subtraction and division by $x^{2^{k-1}}$ can be implemented by logspace-uniform $\mathcal{NC}$ circuits. It remains to implement the divisions on line 5 and line 12 of Algorithm 2, which can be handled by using Lemma B.7. ∎

B.2.3 Final algorithm

Our full algorithm $\mathsf{factorize}_D(Q)$ works as follows:

Algorithm 3 $\mathsf{factorize}_{D,\mathbb{F}}(Q,r,a,t)$

Parameters: D is the maximum degree parameter and $\mathbb{F}$ is a prime field. We require that $|\mathbb{F}| > 2D^2$.

Input: Given $Q \in \mathbb{F}[x,y]$ of degree at most D and $(r,a,t) \in [D] \times \mathbb{F}^2$.

```
1: Compute $Q^{(r-1)} = \frac{\partial^{r-1}Q}{\partial^{r-1}y}$ (where $Q^{(0)} = Q$).
2: Let $Q_a^{(r-1)}(x,y) \leftarrow Q^{(r-1)}(x+a,y)$.
Output:
3: if $\mathsf{PolyLift}_{D,\mathbb{F}}(Q_a^{(r-1)},t) = \perp$ then
4:     return $\perp$
5: else
6:     $\tilde{\tau}(x) \leftarrow \mathsf{PolyLift}_{D,\mathbb{F}}(Q_a^{(r-1)},t)$
7:     return $\tilde{\tau}(x-a)$.
8: end if
```

Theorem B.12 (reminder of Theorem B.5)**.** *Let $\mathbb{F}$ be a prime field and let $D \geq 1$ be an integer such that $|\mathbb{F}| > 2D^2$. Then, there exists an algorithm $\mathsf{factorize}_{D,\mathbb{F}}$ that gets as input $Q \in \mathbb{F}[x,y]$ with degree at most D and advice $(r,a,t) \in [D] \times \mathbb{F}^2$ and satisfies the following:*

1. *For every $\tau(x) \in \mathbb{F}[x]$ such that $(y - \tau(x))$ is a factor of Q, there exists $(r,a,t) \in [D] \times \mathbb{F}^2$ such that $\mathsf{factorize}_{D,\mathbb{F}}(Q, i_{\mathsf{adv}}) = \tau(x)$.*

2. *The algorithm $\mathsf{factorize}_{D,\mathbb{F}}$ can be implemented by logspace-uniform $\mathcal{NC}$ circuits.*

Proof. Let $y - \tau(x)$ be a factor of Q with multiplicity r (note that $1 \leq r \leq \deg(Q) \leq D$). By Lemma B.8 and the assumption on $\mathsf{char}(\mathbb{F})$ (recall that $\mathsf{char}(\mathbb{F}) = |\mathbb{F}|$ since $\mathbb{F}$ is a prime field), it holds that $y - \tau(x)$ is a factor of $Q^{(r-1)} = \frac{\partial^{r-1}Q}{\partial^{r-1}y}$ with multiplicity 1.

Now, by Lemma B.9 and the assumption on $|\mathbb{F}|$, there exists $a \in \mathbb{F}$ such that $(y - \tau(a))$ is a factor of $Q^{(r-1)}(a,y)$ with multiplicity 1. Equivalently, we have that $(y - \tau(0+a))$ is a factor of $Q_a^{(r-1)}(0,y)$ with multiplicity 1. Letting $t = \tau(a)$, by Proposition B.11, it holds that in this case $\mathsf{PolyLift}_{D,\mathbb{F}}(Q_a^{(r-1)},t)$ returns the polynomial $\tau(x+a)$, and $\mathsf{factorize}_{D,\mathbb{F}}(Q,r,a,t)$ reports $\tau(x)$ as desired.

Note that by Proposition B.11, $\mathsf{PolyLift}_{D,\mathbb{F}}(Q_a^{(r-1)}, t)$ can be computed by logspace-uniform NC circuits. Hence, to show the efficiency of $\mathsf{factorize}_{D,\mathbb{F}}$, it suffices to show that in logspace-uniform NC, one can compute $Q_a^{(r-1)}$ from Q, and $\tilde{\tau}(x-a)$ from $\tilde{\tau}(x)$.

Recall that Q is given as a list of coefficients $\{c_{a,b}\}$ such that

$$Q = \sum_{a,b\colon a+b\le D} c_{a,b}x^a y^b.$$

From the rule of computing partial derivative, we have

$$Q^{(r-1)} = \frac{\partial^{r-1}Q}{\partial^{r-1}y} = \sum_{a,b\colon a+b\le D,\ b\ge r-1} c_{a,b}\prod_{j=0}^{r-1}(b-j)\cdot x^a y^{b-r+1}.$$

Since $\prod_{j=0}^{r-1}(b-j)$ can be computed straightforwardly in $O(\log|\mathbb{F}|)$ space, one can compute the coefficient list of $\frac{\partial^{r-1}Q}{\partial^{r-1}y}$ in logspace-uniform NC.

Next, given the coefficient list of $Q^{(r-1)}(x,y)$, we can compute the coefficient list of $Q_a^{(r-1)}(x,y) = Q^{(r-1)}(x+a,y)$ by expanding out every power $(x+a)^t$ in $Q^{(r-1)}(x+a,y)$ and summing everything up, which can be implemented in logspace-uniform $\mathcal{NC}$ by Lemma B.1. Hence one can compute $Q_a^{(r-1)}$ from $Q^{(r-1)}$ by logspace-uniform $\mathcal{NC}$. Similarly, we can also compute $\tilde{\tau}(x-a)$ from $\tilde{\tau}$ by logspace-uniform $\mathcal{NC}$. This concludes the proof. ■

B.3 (Local) List-decoding for polynomial codes

In this section we utilize the factorization algorithm from Theorem B.5 to construct logspace-uniform list decoders for both the Reed-Solomon (RS) codes and the Reed-Muller (RM) codes.

First, in Section B.3.1, we plug Theorem B.5 into the standard list decoding algorithm for the RS code [Sud97], to obtain a logspace-uniform $\mathcal{NC}$ list-decoding algorithm for the RS code. Next, in Section B.3.2, we observe the standard reduction from local list-decoding the RM code to list-decoding the RS code yields a logspace-uniform $\mathcal{NC}$ local list-decoder for the RM code.

B.3.1 List-decoding RS codes in logspace-uniform $\mathcal{NC}$

The following logspace-uniform $\mathcal{NC}$ list-decoding algorithm for the RS code is obtained by using the standard algorithm of [Sud97] with the factorization algorithm from Theorem B.5. We include a proof for completeness, which essentially follows [AB09, Theorem 19.24] while pointing out how the algorithmic steps can be implemented by logspace-uniform randomized $\mathcal{NC}$ circuits.

Theorem B.13 ([Sud97; Sud21]). *Let $\mathbb{F}$ be a prime field. There is an algorithm* $\mathsf{RS\text{-}DecNC}_{\mathbb{F}}$ *that gets as input a set of m pairs $\{(a_i,b_i)\in\mathbb{F}^2\}_{i=1}^m$ and integers $d,t\ge 1$ such that $t > 2\sqrt{dm}$ and $|\mathbb{F}| > 8dm$, and outputs a list of* $\mathrm{poly}(|\mathbb{F}|)$ *polynomials in $\mathbb{F}[x]$ such that the following holds:*

1. *With probability at least $1-2^{-|\mathbb{F}|}$, the output list of* $\mathsf{RS\text{-}DecNC}_{\mathbb{F}}(\{(a_i,b_i)\}_{i=1}^m, d, t)$ *contains every polynomial $\tau\in\mathbb{F}[x]$ of degree at most d satisfying $\Big|\{i\in[m] : \tau(a_i)=b_i\}\Big| \ge t$.*
2. *The algorithm* $\mathsf{RS\text{-}DecNC}_{\mathbb{F}}$ *can be computed by logspace-uniform randomized circuits of size* $\mathrm{poly}(|\mathbb{F}|)$ *and depth* $\mathrm{polylog}(|\mathbb{F}|)$.

Proof. We first find a non-zero polynomial $Q \in \mathbb{F}[x,y]$ with $\deg_x(Q) \leq \sqrt{dm}$ and $\deg_y(Q) \leq \sqrt{m/d}$, such that $Q(a_i, b_i) = 0$ for all $i \in [m]$. This condition can be formulated by m linear equations in the $(\sqrt{dm}+1) \cdot (\sqrt{m/d}+1) > m$ coefficients of Q. Since this linear system is homogeneous and there are more coefficients than equations, one can find a non-zero solution with probability at least $1 - 2^{-|\mathbb{F}|}$ by solving the linear system in randomized logspace-uniform $\mathcal{NC}$, relying on Lemma B.4. The algorithm then enumerates all $\vec{\mathsf{adv}} \in [2\sqrt{dm}] \times \mathbb{F}^2$, adds $\mathsf{factorize}_{2\sqrt{dm},\mathbb{F}}(Q, \vec{\mathsf{adv}})$ to the list if it is not $\perp$, and finally returns the list.

To show the first condition, let $\tau \in \mathbb{F}[x]$ be a polynomial such that $\left| \{i \in [m] : \tau(a_b) = b_i\} \right| \geq t$, and let $\gamma(x) = Q(x, \tau(x))$. Since $\deg_x(Q) \leq \sqrt{dm}$ and $\deg_y(Q) \leq \sqrt{m/d}$ and $\deg(\tau) \leq d$, we have that $\deg(\gamma) \leq 2\sqrt{dm} < t$. On the other hand, by our assumption on τ, for at least t a_i's we have $\gamma(a_i) = Q(a_i, \tau(a_i)) = Q(a_i, b_i) = 0$, which means γ has at least t roots. Since $\deg(\gamma) < t$ it follows that γ is the zero polynomial, and hence $(y - \tau)$ is a factor of Q.

By Theorem B.5, it follows that there exists $\vec{adv} \in [2\sqrt{dm}] \times \mathbb{F}^2$ such that $\mathsf{factorize}_{2\sqrt{dm},\mathbb{F}}(Q, \vec{\mathsf{adv}})$ returns τ, and therefore τ is contained in the list with probability at least $1 - 2^{-|\mathbb{F}|}$. The second condition follows from Theorem B.5, together with the randomized logspace-uniform $\mathcal{NC}$ circuits for solving homogeneous linear systems (from Lemma B.4). ■

B.3.2 Local List-decoding RM codes in logspace-uniform $\mathcal{NC}$

Next we show that local list-decoding the RM code can be done in logspace-uniform $\mathcal{NC}$. The proof of this result implements the reduction from local list-decoding the RM code to list-decoding the RS code by Sudan, Trevisan and Vadhan [STV01] in logspace-uniform $\mathcal{NC}$.

We start by noting that the standard decoder for the RM code is implementable in logspace-uniform $\mathcal{NC}$, and then argue that the *local* list-decoder for the RM code is implementable with such complexity.

Theorem B.14 (decoding the RM code in logspace-uniform $\mathcal{NC}$)**.** *For every prime field $\mathbb{F}$ and integer $\ell \geq 1$ there is a probabilistic procedure $\mathsf{RM\text{-}DecNC}_{\mathbb{F},\ell}$ that gets as input a degree parameter $d \leq |\mathbb{F}|/2$ and a vector $x \in \mathbb{F}^\ell$, and gets oracle access to a function $f\colon \mathbb{F}^\ell \to \mathbb{F}$, and satisfies the following:*

1. *If f agrees with a degree-d polynomial $P\colon \mathbb{F}^\ell \to \mathbb{F}$ on a 0.9 fraction of the inputs, then*

$$\Pr\left[\mathsf{RM\text{-}DecNC}^f_{\mathbb{F},\ell}(d,x) = P(x)\right] \geq 1 - 2^{-2|\mathbb{F}|} .$$

2. *The procedure $\mathsf{RM\text{-}DecNC}_{\mathbb{F},\ell}$ can be implemented by a family of logspace-uniform oracle circuits of size* $\mathrm{poly}(|\mathbb{F}|, \ell)$ *and depth* $\mathrm{polylog}(|\mathbb{F}|)$.

Proof. Recall that the standard decoder for the RM code uniformly chooses $z \in \mathbb{F}^\ell$, queries its oracle f on the input-set $L_{x,z} = \{x + tz : t \in \mathbb{F}\}$, runs the RS list-decoder on the set of pairs $\{(y, f(y)) : y \in L_{x,z}\}$ to obtain a list of univariate polynomials $Q_1, ..., Q_{\mathrm{poly}(|\mathbb{F}|)}\colon \mathbb{F} \to \mathbb{F}$, evaluates each Q_i on $L_{x,z}$, and outputs $Q(0)$ where Q is the polynomial in the list whose agreement with f on $L_{x,z}$ is maximal.

The foregoing procedure can indeed be performed by logspace-uniform probabilistic circuits of size $\mathrm{poly}(|\mathbb{F}|, \ell)$ and depth $\mathrm{polylog}(|\mathbb{F}|)$, relying on Theorem B.13. The standard analysis (see, e.g., [AB09, Proof of Theorem 19.19]) shows that when given access to f that agrees with some degree-d polynomial P on at least $1-(1-d/|\mathbb{F}|)/6 > 0.9$ of inputs, this procedure outputs $P(x)$ with probability at least 0.6. To amplify the success probability we can repeat the process in parallel for $\mathrm{poly}(|\mathbb{F}|)$ times and take the most frequent result. ■

Theorem B.15 (local list-decoding the RM code in logspace-uniform $\mathcal{NC}$)**.** *For every prime field $\mathbb{F}$ and integer $\ell \geq 1$, there is an algorithm $\mathsf{RM\text{-}ListDecNC}_{\mathbb{F},\ell}$ that gets as input a degree parameter $d \geq 1$, a pair $(x_0, y_0) \in \mathbb{F}^\ell \times \mathbb{F}$, and a vector $x \in \mathbb{F}^\ell$, and gets oracle access to a function $f\colon \mathbb{F}^\ell \to \mathbb{F}$, and satisfies the following:*

1. *If f agrees with a degree-d polynomial $P\colon \mathbb{F}^\ell \to \mathbb{F}$ on $10\sqrt{d/|\mathbb{F}|}$ fraction of the inputs, then with probability at least $1/\mathrm{poly}(|\mathbb{F}|)$ over random pairs (x_0, y_0) from $\mathbb{F}^\ell \times \mathbb{F}$,*
$$\Pr[\mathsf{RM\text{-}ListDecNC}^{f}_{\mathbb{F},\ell}(d, x_0, y_0, x) = P(x)] \geq 1 - 2^{-2|\mathbb{F}|} \quad \textit{for every } x \in \mathbb{F}^\ell. \tag{10}$$

2. *$\mathsf{RM\text{-}ListDecNC}_{\mathbb{F},\ell}$ can be implemented by a family of logspace-uniform oracle circuits of size $\mathrm{poly}(|\mathbb{F}|, \ell)$ and depth $\mathrm{polylog}(|\mathbb{F}|, \ell)$.*

Proof. Relying on Theorem B.14, it suffices to describe a relaxed decoder such that (10) holds for at least 0.9 fraction of $x \in \mathbb{F}^\ell$ instead of all possible x (note that we can assume, without loss of generality, that $d \leq |\mathbb{F}|/2$). The decoder of [STV01] works by first constructing a univariate degree-3 curve $q : \mathbb{F} \to \mathbb{F}^\ell$ that passes through x and x_0 (that is, $q(0) = x$ and $q(r) = x_0$ for a random element $r \in \mathbb{F} \setminus \{0\}$), then running the RS list-decoder on the set $\{(t, f(q(t))) : t \in \mathbb{F}\}$ to obtain a list $g_1, ..., g_{\mathrm{poly}(|\mathbb{F}|)}$ of polynomials, and finally evaluating all g_i's on a given point, comparing the results to see if there is a unique g_i such that $g_i(r) = y_0$, and if so outputs $g_i(0)$ (see [AB09, Theorem 19.26] for a detailed description and proof of correctness).

The RS list-decoder from Theorem B.13 is in logspace-uniform $\mathcal{NC}$, and thus it suffices to show that the curve q can be constructed by log-space uniform oracle circuit of size $\mathrm{poly}(|\mathbb{F}|, \ell)$ and depth $\mathrm{polylog}(|\mathbb{F}|, \ell)$. To do this the decoder fixes two distinct elements $u, v \in \mathbb{F} \setminus \{0, r\}$, draws two uniformly random vectors x_1 and x_2 from $\mathbb{F}^\ell$, and writes a linear system over 4ℓ variables (the coefficients of q) expressing the four conditions $q(0) = x$, $q(r) = x_0$, $q(u) = x_1$, and $q(v) = x_2$. This linear system has a unique solution, and thus by Lemma B.3 it can be solved in logspace-uniform $\mathcal{NC}$. ■

B.4 Sample-aided worst-case to rare-case reductions for polynomials

Proposition B.16 (low-degree polynomials are uniformly sample-aided worst-case to rare-case reducible; Proposition 3.10, restated)**.** *Let $q\colon \mathbb{N} \to \mathbb{N}$ be a function mapping integers to primes, let $\ell\colon \mathbb{N} \to \mathbb{N}$ such that $n \geq \ell(n) \cdot \log(q(n))$, and let $d\colon \mathbb{N} \to \mathbb{N}$. Let $f = \{f_n\}_{n \in \mathbb{N}}$ be a sequence of functions such that f_n computes a polynomial $\mathbb{F}_n^{\ell(n)} \to \mathbb{F}_n$ of degree $d(n)$ where $|\mathbb{F}_n| = q(n)$. Then f is sample-aided worst-case to ρ-rare-case reducible by logspace-uniform oracle circuits of size* $\mathrm{poly}(q, \ell)$ *and depth* $\mathrm{polylog}(q, \ell)$ *with error $1 - 2^{-q}$ and* $\mathrm{poly}(q)$ *samples, where $\rho = 10\sqrt{d(n)/q(n)}$.*

Proof. We construct a logspace-uniform probabilistic oracle circuit M that gets input 1^n, and oracle access to a function $\widetilde{f_n}$ that agrees with f_n on a $\rho(n)$ fraction of the inputs, and also

$\mathrm{poly}(1/\rho)$ labeled samples for f_n, and with probability $1-2^{-q}$ outputs a circuit $C\colon \mathbb{F}^{\ell} \to \mathbb{F}$ such that for every $x \in \mathbb{F}^{\ell}$ it holds that $\Pr_r[C^{\widetilde{f_n}}(x,r) = f_n(x)] \geq 2/3$. Specifically, M works as follows:

1. The first step is to construct in parallel $t = \mathrm{poly}(q)$ logspace-uniform circuits such that each circuit implements the algorithm from Theorem B.15 with an independent uniform choice of (x_0, y_0) (and with degree parameter $d = d(n)$).

   This yields a list of t probabilistic oracle circuits $C_1, ..., C_t$ such that with probability at least $1-2^{-2q}$, there exists $i \in [t]$ for which $\Pr[C_i^{\widetilde{f_n}}(x) = f_n(x)] \geq 1-2^{-2q}$ for all $x \in \{0,1\}^n$. We call any circuit that satisfies the latter condition good.

2. The second step is to choose random coins for each circuit C_i and hard-wire them, transforming C_i into a deterministic circuit. Specifically, for each C_i we independently try $w = \mathrm{poly}(q)$ different random strings, obtaining a set of w deterministic circuits $C_{i,1}, ..., C_{i,w}$. This yields $t' = t \cdot w = \mathrm{poly}(q)$ deterministic circuits $D_1, ..., D_{t'}$ such that with probability $1-2^{-2q}$ there exists a circuit D_i satisfying $\Pr_x[D_i^{\widetilde{f_n}}(x) = f_n(x)] \geq 0.99$. [46]

3. The third step is to "weed" the list in order to find a single circuit D_i that (when given access to $\widetilde{f_n}$) correctly computes f_n on 0.95 of the inputs. To do so we iterate over the list in parallel, and for each circuit D_j we estimate the agreement of $D_j^{\widetilde{f_n}}$ with f_n with accuracy 0.01 and confidence $1-2^{-2q}$, using $\mathrm{poly}(q)$ random samples.

4. The final step is to compose D_i with the decoder from Theorem B.14 to obtain a probabilistic circuit that correctly computes f at each input with probability at least $1-2^{-2q}$.

The success probability of the foregoing procedure is $1-O(2^{-2q}) > 1-2^{-q}$. We now argue that the foregoing procedure M can be implemented as a logspace-uniform probabilistic circuit of size $\mathrm{poly}(q,\ell)$ and depth $\mathrm{polylog}(q,\ell)$; for simplicity, below we just state that circuits are "logspace-uniform", and by this we implicitly mean circuits of such size and depth.

Note that in the first step we just need to compute the description of $t = \mathrm{poly}(q)$ circuits $C_1, ..., C_t$, whose random coins are the ones chosen by our logspace-uniform circuit M. Since each C_i is logspace-uniform, the part of M that computes their descriptions is also logspace-uniform. The second step is computationally very easy given the descriptions of the C_i's. The third step amounts to simulating the circuits whose descriptions were already computed, so this step can indeed be executed by a logspace-uniform circuit. And the fourth step is dominated by the cost of computing the description of a fixed logspace-uniform circuit, and replacing its oracle gates by the descriptions of the circuit D_i found in the previous step. ■

Appendix C An unconditional approximate-direct-product result

We include for completeness a proof of Proposition 6.10. As mentioned in Section 6, the proof follows from the results of Impagliazzo *et al.* [IJK+10], with one caveat being that we refer to

[46] This success bound assumes that $\mathbb{F}$ is sufficiently large such that the success probability of a good C_i satisfies $1-2^{-2q} > .999$. If $\mathbb{F}$ is too small then we can amplify the success probability of each C_i in the first step by implementing naive error reduction.

standard direct-product functions, whereas their direct-product function $f^{\times k}(x_1, ..., x_k)$ only takes *distinct* inputs $x_1, ..., x_k$ (i.e., it disallows $x_i = x_j$ for any $i \neq j \in [k]$).

Proposition C.1 (non-strong approximate-direct-product theorem)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $\delta \in (0, 1/2)$, let $\alpha > 0$ be sufficiently small, let $T \colon \mathbb{N} \to \mathbb{N}$ be time-computable, and let $k(n) = o(n)$. Then, for any $f \notin \mathsf{avg}_{1-\delta}\text{-}\mathcal{BPTIME}[T]//(1/\alpha)$ and any probabilistic algorithm A that runs in time $T(n) - n^c$, the probability over $z \in \{0,1\}^{k \cdot n}$ that A approximately-prints $f^{\times k}(z)$ with error α is at most δ.*

Proof. Given $\delta > 0$, let $\delta' = \delta/12$, let $\gamma < \delta'$ be a sufficiently small constant and let $\alpha < \gamma$ be a sufficiently small constant. We will assume that $k(n)$ is larger than $\delta'/2\gamma$. Assume towards a contradiction that an algorithm A as in the statement exists, and let $Z \subseteq \{0,1\}^{k \cdot n}$ be the set of inputs z such that with probability at least $1 - \alpha$ we have $\Pr_{i \in [k]}[A(z)_i = g(z)_i] \geq 1 - \alpha$. We denote by $A(z, r)$ the decision of A on input z with random coins r.

Choosing a set $S \subseteq [k]$. The first step of our algorithm is to randomly choose a set $S \subseteq [k]$ of size $\delta'/4\gamma < k/2$. Our hope is that there are many inputs z on which A, with high probability over internal choice of random coins, correctly computes *all the coordinates in S* of $g(z)$; in other words, we hope that there are many z's such that A, with high probability over coins, correctly computes f on *all* the inputs $(z_i)_{i \in S}$. For any input z, and for a fixed $i \in [k]$, we denote $\mu_i(z) = \Pr_r[A(z,r)_i \neq g(z)_i]$; similarly, for a fixed $S \subseteq [k]$ we denote $\mu_S(z) = \Pr_r[A(z,r)_S \neq g(z)_S]$. Then, we claim that:

Claim C.1.1. *With probability more than $1 - \delta'$ over choice of S, there exists a set Z' of size at least $|Z|/2$ such that for every $z \in Z'$ we have that $\mu_S(z) \leq 2\alpha/\gamma$.*

Proof. For every fixed $z = (x_1, ..., x_k) \in Z$ we have that $\mathbb{E}_{i \in [k]}[\mu_i(z)] = \Pr_{i \in [k], r}[A(z,r)_i \neq g(x)_i] \leq 2\alpha$, and hence

$$\Pr_{i \in [k]}[\mu_i(z) \geq 2\alpha/\gamma] \leq \gamma \ . \tag{C.1}$$

By Eq. (C.1), for every fixed $z \in Z$, when choosing the first element $i_1 \in [k]$ for S, the probability that $\mu_{i_1}(z) < 2\alpha/\gamma$ is at most γ. When choosing the second elements i_2, the probability that $\mu_{i_2}(z) \geq 2\alpha/\gamma$ is at most $\frac{\gamma \cdot k}{k-1}$; by induction, when adding each element i_j, the probability that $\mu_{i_j}(z) \geq 2\alpha/\gamma$ is less than $\frac{\gamma \cdot k}{k - |S|}$. Since k is sufficiently large, we can bound this probability by $\frac{\gamma \cdot k}{k-|S|} < \frac{\gamma \cdot k}{k/2} = 2\gamma$, where we used the fact that $|S| < k/2$. By a union-bound, the probability over S that $\mu_S(z) \geq 2\alpha/\gamma$ is at most $2\gamma|S| < \delta'/2$.

For every fixed $z \in Z$, and for a random choice of S, denote the event that $\mu_S(z) < 2\alpha/\gamma$ by $\mathcal{G}(z)$. The above shows that for every $z \in Z$ we have that $\Pr_S[\neg\mathcal{G}(z)] < \delta'/2$. It follows that $\mathbb{E}_S[\Pr_{z \in Z}[\neg\mathcal{G}(z)]] < \delta'/2$, and hence the probability over S that $\Pr_{z \in Z}[\neg\mathcal{G}(z)] < 1/2$ is at most δ'. □

Choosing fixed random coins r. As a second step our algorithm will hard-wire random coins r into A, yielding a deterministic algorithm $A_r(z) = A(z, r)$. We claim that with probability at least $1-\delta'$ over choice of random coins r, there exists a set $Z'' \subseteq Z'$ of density $1-\frac{2\alpha}{\gamma\cdot\delta'} > 1/2$ in Z' such that for every $z \in Z''$ we have that $A_r(z)_S = g(z)_S$. To see this, note that

$$\mathbb{E}_r\left[\Pr_{z\in Z'}[A(z,r)_S \neq g(z)_S]\right] = \mathbb{E}_{z\in Z'}\left[\Pr_r[A(z,r)_S \neq g(z)_S]\right] \leq 2\alpha/\gamma\ ,$$

and hence the probability over r that $\Pr_{z\in Z'}[A(z,r)_S \neq g(z)_S] > \frac{2\alpha}{\gamma\cdot\delta'}$ is at most δ'.

Invoking the algorithm of [IJK+10] with the fixed S and r. For $m = |S|$, let $h = f^{\times m}$ be the m-wise direct-product of f. We now use the following algorithm by Impagliazzo *et al.* [IJK+10]:

Theorem C.2 (uniform list-decoding of the direct-product code)**.** *There is a constant $c_{\mathsf{IJKW}} > 1$ and a probabilistic algorithm* `Dec` *such that for any $f\colon \{0,1\}^n \to \{0,1\}$ and $m \in \mathbb{N}$ and $\epsilon, \delta \in (0,1)$ satisfying $\epsilon > e^{-\delta m/c_{\mathsf{IJKW}}}$ the following holds. Let $h = f^{\times m}$. Then, when* `Dec` *is given oracle access to a function $\widetilde{h}\colon \{0,1\}^{m\cdot n} \to \{0,1\}^m$ satisfying $\Pr_{z\in\{0,1\}^{m\cdot n}}[\widetilde{h}(z) = h(z)] \geq \epsilon$, with probability $\Omega(\epsilon)$ it outputs an oracle circuit $C\colon \{0,1\}^n \to \{0,1\}$ such that $\Pr_{x\in\{0,1\}^n}[C^{\widetilde{h}}(x) = f(x)] \geq 1-\delta$. Moreover, the algorithm* `Dec` *is a uniform randomized $\mathcal{NC}^0$ algorithm that makes just one oracle query to $\widetilde{h}$, and the circuit C is an $\mathcal{AC}^0$ circuit of size $\mathrm{poly}(n, m, \log(1/\delta), 1/\epsilon)$ with $O(\log(1/\delta)/\epsilon)$ oracle gates to $\widetilde{h}$.*

We invoke the algorithm `Dec` from Theorem C.2 with parameter values $\epsilon = \delta/4$ and δ' and m, while giving it oracle access to the function $\widetilde{h}(x_1, ..., x_m) = A_r(z)_{S_1}, ..., A_r(z)_{S_m}$. The constraint on the parameters in Theorem C.2 is

$$\epsilon > e^{-\delta' m/c_{\mathsf{IJKW}}} \iff \delta/4 > e^{-(\delta')^2/(4\gamma\cdot c_{\mathsf{IJKW}})}\ ,$$

where the " $\iff$ " is since $m = |S| = \delta'/4\gamma$. The constraint above is then satisfied due to our choice of a sufficiently small γ.

With probability at least $\Omega(\delta)$ the algorithm outputs an oracle circuit that computes f on $1-\delta'$ of the inputs when given access to A_r. Thus, if we run this algorithm for $O((1/\delta)\cdot\log(1/\delta'))$ times, with probability at least $1-\delta'$ at least one of the resulting circuits will compute f correctly on $1-\delta'$ of the inputs when given access to A_r. We get as advice an index for the circuit with maximal agreement with f, which we denote from now on by C. The size of C is at most $\mathrm{poly}(n, m, \log(1/\delta'), 1/\epsilon) = \mathrm{poly}(n)$, and it makes at most $O(\log(1/\delta')/\epsilon) = O(1)$ queries to A_r.

The final algorithm. Our algorithm constructs C as above, which is a procedure that does not depend on the input $x \in \{0,1\}^n$, and then outputs $C(x)$. We claim that there exists a set $X \subset \{0,1\}^n$ of density $1-9\delta' = 1-\delta$ such that for every $x \in X$ we have that $\Pr_C[C(x) = f(x)] \geq 2/3$. To see this, recall that the probability that all the three steps in the construction of C above succeed (i.e., we choose a good S, choose a good r, and the invocations of the algorithm from Theorem C.2 succeed) is at least $1-3\delta'$. Whenever that happens, the circuit C correctly computes f on $1-\delta'$ of the inputs x. Thus, we have that

$$\mathbb{E}_x\left[\Pr_C[C(x) \neq f(x)]\right] = \mathbb{E}_C\left[\Pr_x[C(x) \neq f(x)]\right] \leq 4\delta'\ ,$$

and hence the probability over x that $\Pr[C(x) \neq f(x)] \geq 1/3$ is at most $12\delta' = \delta$.

Indeed, the algorithm above is probabilistic, and on at least $1-\delta$ of the inputs computes f with probability at least $2/3$. The running time of the algorithm is dominated by the step of evaluating the oracle circuit C using the algorithm A, which can be done in time $\mathrm{poly}(n) + O(T'(n)) < T(n)$. Also, the algorithm relies on $O(\log(1/\delta))$ bits of advice that depend only on the randomness and not on the input. This contradicts our assumption that $f \notin \mathsf{avg}_{1-\delta}\text{-}\mathcal{BPTIME}[T]//(1/\alpha)$. ■

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il