Electronic Colloquium on Computational Complexity, Report No. 10 (2025)

Towards Free Lunch Derandomization from Necessary Assumptions (and OWFs)

Marshall Ball* Lijie Chen† Roei Tell‡

Abstract

The question of optimal derandomization, introduced by Doron et. al (JACM 2022), garnered significant recent attention. Works in recent years showed conditional superfast derandomization algorithms, as well as conditional impossibility results, and barriers for obtaining superfast derandomization using certain black-box techniques.

Of particular interest is the extreme high-end, which focuses on "free lunch" derandomization, as suggested by Chen and Tell (FOCS 2021). This is derandomization that incurs essentially no time overhead, and errs only on inputs that are infeasible to find. Constructing such algorithms is challenging, and so far there have not been any results following the one in their initial work. In their result, their algorithm is essentially the classical Nisan-Wigderson generator, and they relied on an ad-hoc assumption asserting the existence of a function that is non-batch-computable over all polynomial-time samplable distributions.

In this work we deduce free lunch derandomization from a variety of natural hardness assumptions. In particular, we do not resort to non-batch-computability, and the common denominator for all of our assumptions is hardness over all polynomial-time samplable distributions, which is necessary for the conclusion. The main technical components in our proofs are constructions of new and superfast targeted generators, which completely eliminate the time overheads that are inherent to all previously known constructions. In particular, we present an alternative construction for the targeted generator by Chen and Tell (FOCS 2021), which is faster than the original construction, and also more natural and technically intuitive.

These contributions significantly strengthen the evidence for the possibility of free lunch derandomization, distill the required assumptions for such a result, and provide the first set of dedicated technical tools that are useful for studying the question.

*New York University. Email: marshall@cs.nyu.edu

†Miller Institute for Basic Research in Science, University of California, Berkeley. Email: wjmzmbr@gmail.com

‡University of Toronto, Department of Computer Science. Email: roei@cs.toronto.edu

ISSN 1433-8092

Contents

1 Introduction

The classical $prBPP = prP$ conjecture asserts that all randomized algorithms solving decision problems can be efficiently simulated by deterministic algorithms, with only a polynomial time overhead. Celebrated hardness vs randomness results in complexity theory conditionally yield such derandomization, while incurring an overhead that is polynomial but large [IW97; KM02; SU05; Uma03]. Can we go further than that, simulating randomness with smaller overhead?

The challenge of finding the minimal overhead for derandomization was introduced by Doron *et al.* [DMO+20], and has been intensively studied since. The goal in this context is to construct "superfast" derandomization algorithms, with small overhead and ideally with no overhead at all, under the weakest possible hardness assumption. Among the known results are conditional superfast derandomization algorithms (see, e.g., [DMO+20; CT21b; CT21a]), conditional impossibility results (see [CT21a; CT23b]), barriers for certain black-box techniques (see [SV22]), and a study of this question in the setting of interactive proof systems (see [CT23b]) and in the space-bounded setting (see [DT23; DPT24]).

When focusing on superfast derandomization that succeeds on all inputs (i.e., in the worst-case), a potential emerging picture is starting to seem clearer. Following [DMO+20], Chen and Tell [CT21b] showed that assuming one-way functions and sufficiently strong lower bounds for non-uniform procedures (i.e., for algorithms with advice), we can simulate all randomized time-T algorithms in deterministic time $n \cdot T^{1+\epsilon}$. They also showed that, assuming #NSETH (i.e., a strong assumption from the exponential-time hypotheses family, see [CGI+16]), this is optimal: there is no worst-case derandomization running in time $n \cdot T^{1-\epsilon}$.

Unfortunately, for fast randomized algorithms (e.g., running in linear time), this overhead is significant. Is this indeed the price we must pay for derandomization?

"Free lunch" derandomization. A subsequent work of Chen and Tell [CT21a] offered a way out: Following Impagliazzo and Wigderson [IW98], they considered derandomization that succeeds not in the worst-case, but over *all polynomial-time samplable distributions*. That is, the deterministic simulation errs on some inputs, but these inputs are infeasible to find. In other words, such derandomization appears correct to every efficient observer.

Definition 1.1 (heuristic simulation)**.** *For $L \subseteq \{0,1\}^*$ and a class $\mathcal{C}$ of languages, we say that $L \in$ heur-$\mathcal{C}$ if there is $L' \in \mathcal{C}$ such that for every probabilistic polynomial-time algorithm F it holds that $\Pr_{x \leftarrow F(1^n)}[x \in \Delta(L, L') \cap \{0,1\}^n] \leq n^{-\omega(1)}$. The definition extends to the more general notion of promise-problems $\Pi = (\Pi^Y, \Pi^N) \in$ heur-prC in the natural way (see Definition 3.2).*

Constructing free lunch derandomization algorithms is a challenging problem, and at the moment we only have one conditional construction (see below). We point out two concrete technical obstacles that have so far hindered progress:

1. **The "hybrid argument" challenge.** [1] Assume that we just want to reduce the number of coins of a probabilistic polynomial-time algorithm from $\text{poly}(n)$ to n^ϵ. Almost all known ways to do this incur a polynomial time overhead, due to the "hybrid argument" challenge, which has been studied for decades (see, e.g., [SV22; LPT24], and the references therein).

   The two known ways to handle this barrier either assume one-way functions [CT21b], or assume hardness against non-uniform Merlin-Arthur circuits [DMO+20]. We are not aware

[1] This is usually referred to as the "hybrid argument barrier", and we replace "barrier" with "challenge" to make the point that this challenge should be tackled.

of any reason to suspect that either of these assumptions is necessary to deduce the derandomization outcome (even just "morally" or qualitatively, let alone quantitatively).

2. **The lack of technical tools to exponentially reduce the number of coins (with no overhead).** Assume that we somehow handled the hybrid argument challenge, and every probabilistic polynomial-time algorithm now uses only n^ϵ random coins. Even then, when instantiating the known hardness-vs-randomness tools with standard assumptions (e.g., circuit lower bounds, or hardness for standard probabilistic algorithms), we incur significant runtime overheads. In particular, the many constructions of targeted pseudorandom generators and hitting set-generators from recent years incur such runtime overheads (see Section 2.1).

   The only previously known way to bypass this falls back to using classical technical tools (i.e., it simply used the Nisan-Wigderson generator [NW94]), instantiated with non-standard assumptions. Specifically, the single previously known result relied on an ad-hoc assumption, which – yet again – we have no reason to believe is necessary for the conclusion.

The aforementioned previously known result, from the original work of Chen and Tell [CT21a], deduced that $\mathcal{BPTIME}[T] \subseteq \mathsf{heur}\text{-}\mathcal{DTIME}[T^{1+O(\epsilon)}]$ relying on one-way functions (to bypass the hybrid argument challenge), and on the following assumption: There is a function $f\colon \{0,1\}^n \to \{0,1\}^{n^\epsilon}$ such that $f(x)$ is hard to *approximately batch-compute* when x is sampled from *any polynomial-time samplable distribution*.[2] In an analogous result for the non-deterministic setting, a subsequent work of Chen and Tell [CT23b] deduced free lunch derandomization of certain classes of interactive proof systems into deterministic $\mathcal{NP}$-type protocols, under stronger assumptions of a similar "non-batch-computability" flavor (see [CT23b] for precise details).

1.1 Our contribution, at a high level

In this work we deduce free lunch derandomization from a variety of natural hardness assumptions. The common denominator for all of our assumptions is hardness over all polynomial-time samplable distributions, which is necessary for the conclusion (see Claim 3.4). This significantly strengthens the evidence for the possibility of free lunch derandomization, both since our assumptions are more standard and well-studied (compared to non-batch-computability), and since the conclusion is now known to follow from various different natural assumptions.

Our main technical contribution is a dedicated set of technical tools for studying the problem of free lunch derandomization, addressing the second technical obstacle among the two obstacles mentioned above. Specifically, we construct targeted pseudorandom generators that are superfast, and that completely avoid the large polynomial time overheads that were inherent to many recent constructions. Our most interesting technical contribution is an alternative construction for the targeted generator of Chen and Tell [CT21a] (i.e., their generator that was originally used to deduce $pr\mathcal{BPP} = pr\mathcal{P}$, rather than free lunch derandomization), which is considerably faster than the original construction, and also more natural and technically intuitive.

We view our work as the first technical step following [CT21a] towards deducing free lunch derandomization from necessary assumptions. As part of this effort, we also show that non-batch-computability as in [CT21a; CT23b] (by itself, i.e. not necessarily over all polynomial-time samplable distributions) is not rare or hard to get: we deduce it from standard worst-case

[2] That is, each output coordinate $f(x)_i$ is computable in time $T^{1+\epsilon}$, but for every time-T algorithm A, and every polynomial-time samplable distribution over inputs x, with all but negligible probability over x it holds that $A(x)$ does not print all of $f(x)$ in time significantly faster than $T \cdot |f(x)|$ (i.e., in time $T \cdot |f(x)|^\epsilon$). (In fact, they assumed that $A(x)$ cannot even print an approximate version of $f(x)$; we ignore this for simplicity of exposition.)

hardness assumptions. This gives further evidence that the non-batch-computability assumption was ad-hoc, whereas the main key is hardness over all polynomial-time samplable distributions.

A preliminary observation about a more relaxed notion. A more relaxed notion than the one in Definition 1.1 was considered in several prior works, which considered derandomization that has essentially no time overhead and that succeeds on average over the *uniform distribution*. Previous works deduced such derandomization from OWFs and additional hardness assumptions (see [CT21b; CT21a]). We observe that such derandomization actually follows from OWFs *alone*, without additional assumptions, as a consequence of a lemma by Klivans, van Melkebeek, and Shaltiel [KMS12] (see Section 3.1.2). Thus, throughout the paper we focus on the stronger notion of free lunch derandomization from Definition 1.1, wherein errors are infeasible to find.

Assuming one-way functions. As in previous works, our results assume the existence of one-way functions. Since this is a ubiquitous and widely believed assumption, it strikes us as more urgent to improve the ad-hoc non-batch-computability assumption from [CT21a; CT23b] than to remove the OWF assumption. We stress that for free lunch derandomization (i.e., as in Definition 1.1), the standard OWFs we assume (secure against polynomial-time adversaries) do not seem to yield anything better than subexponential-time derandomization.[3] Moreover, our results actually only need a weaker assumption, which distills what is actually used in OWFs; for details and further discussion, see Assumption 3.7, Assumption 3.9 and Section 1.2.2.

1.2 Free lunch derandomization from standard hardness

Our first result obtains superfast derandomization, as well as free lunch derandomization with a small amount of advice bits (i.e., $\tilde{O}(\log n)$) from hardness over all polynomial-time samplable distributions of functions computable in *bounded depth* $n^{o(1)}$. That is:

Theorem 1.2 (free lunch derandomization from hardness of $n^{o(1)}$-depth circuits; informal, see Theorem 5.2)**.** *Assume that OWFs exist, and that for every polynomial $T(n)$ there is $f\colon \{0,1\}^* \to \{0,1\}^*$ computable by sufficiently uniform circuits of size $T^{1+\epsilon}$ and depth $n^{o(1)}$ that is hard for probabilistic algorithms running in time $T^{1-\epsilon}$ over all polynomial-time samplable distributions. Then, for every polynomial $T(n)$ we have that $prBPTIME[T] \subseteq \mathsf{heur}\text{-}prDTIME[T^{2+O(\epsilon)}]$, and*

$$prBPTIME[T] \subseteq \mathsf{heur}\text{-}prDTIME[T^{1+O(\epsilon)}]/\tilde{O}(\log n)\ .$$

The $\tilde{O}(\log n)$ bits of advice can be removed, at the cost of strengthening the OWFs assumption to a more general one of a similar flavor; see Section 1.2.2 for details. The crux of Theorem 1.2 is that we only assume standard hardness for functions computable in $n^{o(1)}$-depth, which is a more standard and well-studied assumption than non-batch-computability.

The proof of Theorem 1.2 is the main technical contribution of this work. Indeed, while the assumption in Theorem 1.2 is reminiscent of the ones used to deduce $prBPP = prP$ in [CT21a], we do not use their technical tools. Loosely speaking, we construct a new targeted generator based on functions computable in bounded depth such that the generator has almost no time overhead. This construction can be viewed as a variant of the generator from [CT21a], which was also based on functions computable in bounded depth. The main new insight underlying our construction is that we can replace the doubly efficient proof systems of Goldwasser, Kalai, and

[3] Alternatively, standard OWFs can yield free lunch derandomization with n^{ϵ} bits of advice (see Section 2.1.3).

Rothblum [GKR15] (which were used in the previous construction) with PCP-based techniques, and in particular relying on arithmetizations of highly efficient sorting networks.

This construction is of independent interest (i.e., beyond free lunch derandomization). As mentioned above, it is more natural and technically intuitive than the previous one, while relying on very similar assumptions regarding the hard function. Thus, it can serve as a general alternative to the known proof-system-based targeted PRGs. See Section 2.1 for details.

1.2.1 Free lunch derandomization from two other assumptions

We are also able to deduce the same conclusion as in Theorem 1.2 from other natural assumptions. For example, using the new targeted generator mentioned above, we can deduce the same conclusion from the assumption that there are functions computable in time T but not in space $T^{1-\epsilon}$ and time $T^{1+\epsilon}$, where hardness is over all polynomial-time samplable distributions.

Theorem 1.3 (free lunch derandomization from time-space tradeoffs; informal, see Theorem 5.4)**.** *Assume that OWFs exist, and that for every polynomial $T(n)$ there is $f\colon \{0,1\}^* \to \{0,1\}^*$ computable in time T that is hard for probabilistic algorithms running in space $T^{1-\epsilon}$ and time $T^{1+\epsilon}$ over all polynomial-time samplable distributions. Then, for every polynomial $T(n)$ we have $prBPTIME[T] \subseteq$ heur-$prDTIME[T^{2+O(\epsilon)}]$, and $prBPTIME[T] \subseteq$ heur-$prDTIME[T^{1+O(\epsilon)}]/\tilde{O}(\log n)$.*

In yet another variation on the assumptions, we also show that free lunch derandomization follows from the existence of a function $x \mapsto f(x)$ such that $f(x)$ is hard to *efficiently learn* (from membership queries) when x comes from any polynomial-time samplable distribution.

Theorem 1.4 (free lunch derandomization from hardness of learning; informal, see Theorem 5.7)**.** *Assume that OWFs exist, let T be a polynomial, and assume that there is $f\colon \{0,1\}^n \to \{0,1\}^{k=n^\epsilon}$ computable in time $T^{1+O(\epsilon)}$ such that for every probabilistic algorithm M running in time $T^{1+\epsilon}$, when sampling x from any polynomial-time samplable distribution, with all but negligible probability M fails to learn $f(x)$ with accuracy 0.99 from $k^{.01}$ membership queries. Then, $prBPTIME[T] \subseteq$ heur-$prDTIME[T^{1+O(\epsilon)}]$.*

In contrast to Theorem 1.3, the proof of Theorem 1.4 does not leverage our new technical tools, and instead relies on ideas similar to the ones of Liu and Pass [LP22a; LP22b], and uses the Nisan-Wigderson generator. Similarly to [LP22a; LP22b], we can actually obtain an equivalence between the conclusion and the hardness hypothesis, assuming OWFs; see Theorem 5.7 for details.

The point in stating the variations above is to demonstrate that several different hardness assumptions suffice for free lunch derandomization, the only common feature of which is hardness over all polynomial-time samplable distributions.

1.2.2 What is actually being used in OWFs? Removing the $\tilde{O}(\log n)$ advice bits

If one is willing to generalize the OWF assumption, then the $\tilde{O}(\log n)$ bits of advice can be removed. Specifically, we introduce a natural assumption that distills the actual content of the OWF assumption that is being used in the proofs and generalizes it, and show that replacing OWFs by this more general assumption suffices to eliminate the advice.

In our proofs and in [CT21b; CT21a], the OWF is only used to obtain a PRG that is computable in near-linear-time $n^{1+\epsilon}$, has polynomial stretch $n^\epsilon \mapsto n$, and fools linear-time algorithms.[4] This is spelled out in Assumption 3.7. We observe that the crucial parameters in the foregoing assumption are: (1) The near-linear running time of the PRG; (2) The lower running time of the

[4] We stress that the "cryptographic" properties of this PRG are not used, in the sense that we only need the PRG to fool algorithms that run in less time than the PRG itself.

distinguishers; (3) The polynomial stretch. What does *not* seem like a key defining property, however, is that the polynomial stretch is $n^\epsilon \mapsto n$, rather than $\ell^\epsilon \mapsto \ell$ for some $\ell \ll n$. Hence, a more general version of the foregoing assumption asserts that there is a PRG running in time $n^{1+\epsilon}$, fooling $O(n)$-time adversaries, and stretching ℓ^ϵ bits to ℓ bits, where ℓ may be smaller than n. This general version is spelled out in Assumption 3.9, and may be of independent interest.

The more general assumption, by itself, does not seem to yield anything better than subexponential-time derandomization (as one may expect, given that the PRG only has polynomial stretch), and certainly does not seem to imply any cryptographic primitive. However, we prove that if we replace OWFs by this assumption in Theorems 1.2 and 1.3, we can deduce free lunch derandomization *without non-uniform advice*. For details, see Section 2.1 and Theorems 5.1 and 5.3.

1.3 Free lunch derandomization in the non-deterministic setting

In the non-deterministic setting, the situation gets quite better. In this setting, we are interested either in derandomizing probabilistic algorithms non-deterministically (i.e., in showing superfast versions of $\mathcal{BPP} \subseteq \mathcal{NP}$) or in derandomizing Merlin-Arthur protocols non-deterministically (i.e., superfast versions of $\mathcal{MA} \subseteq \mathcal{NP}$). For concreteness, let us focus on the latter.

For context, recall that any worst-case derandomization of $\mathcal{MA}$ incurs quadratic overhead, assuming #NSETH (i.e., $\mathcal{MATIME}[T] \not\subseteq \mathcal{NTIME}[T^{2-\epsilon}]$ for any $\epsilon > 0$; see [CT23b, Theorem 6.1]). Hence, in this context too we focus on derandomization in which errors may exist but are infeasible to find. Specifically, recalling that derandomization of an $\mathcal{MA}$ verifier is conducted with respect to both an input x and a witness π (that are given to the verifier), we are interested in a notion in which it is infeasible to find a pair (x, π) that causes a derandomization error.

Following [CT23b], we consider free lunch derandomization of $\mathcal{MA}$ into computationally sound $\mathcal{NP}$ protocols. A cs-$\mathcal{NTIME}[T]$ protocol consists of a *deterministic* time-T verifier V and an efficient prover P such that P can prove every correct statement to V (i.e., by sending a static, $\mathcal{NP}$-style witness π), yet no efficient uniform adversary can find an input x and proof π that mislead V, except with negligible probability.[5] Indeed, a cs-$\mathcal{NP}$ protocol is a deterministic argument system; see Definition 6.1 for formal details, and for further background on this class see [CT23b] and the very recent work of Chen, Rothblum, and Tell [CRT25].

The basic result: Free lunch derandomization from hardness in $\mathcal{FP}$. Our first result is free lunch derandomization of $\mathcal{MA}$ into cs-$\mathcal{NP}$ relying on a surprisingly clean assumption: we just need a function computable in time $n^{1+O(\epsilon)}$ that is hard for $\mathcal{MATIME}[n^{1+\epsilon}]$ over all polynomial-time samplable distributions. Indeed, this assumption does not involve non-batch-computability, low-depth, time-space tradeoffs, or any other non-standard property.

Theorem 1.5 (free lunch derandomization of $\mathcal{MA}$; informal, see Theorem 6.4)**.** *Assume that OWFs exist, and that there is $f\colon \{0,1\}^n \to \{0,1\}^{n^{o(1)}}$ computable in time $n^{1+O(\epsilon)}$ that is hard for $\mathcal{MATIME}[n^{1+\epsilon}]$ over all polynomial-time samplable distributions. Then, for every polynomial $T(n)$,*

$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{1+\epsilon}]/\tilde{O}(\log n)\ .$$

The hypothesis in Theorem 1.5 is reminiscent of the one in [DMO+20], but they deduce worst-case derandomization with quadratic overhead (of probabilistic algorithms), whereas Theorem 1.5 deduces free lunch derandomization with essentially no overhead (of $\mathcal{MA}$ into cs-$\mathcal{NP}$

[5] When considering cs-$\mathcal{NP}$ protocols for $L \in \mathcal{MA}$, we equip the honest prover with a witness in a witness-relation for L (otherwise the prover cannot be efficient). This is the standard practice when defining argument systems in cryptography (see, e.g., [Gol01, Section 4.8]), and it extends to deterministic argument systems; see [CRT25].

protocols). Indeed, we do not use their technical tools. We prove Theorem 1.5 using a superfast targeted generator that is suitable for the non-deterministic setting, and which is a variant of a PCP-based generator recently introduced by van Melkebeek and Sdroievski [MMS23]. For details, see Section 2.2. Similarly to Section 1.2.2, we can remove the $\tilde{O}(\log n)$ bits of advice by replacing the OWFs assumption with an assumption about the existence of a near-linear-time PRG for a general setting of parameters (see Section 6 for details).

The conclusion in Theorem 1.5 addresses a significantly more general setting compared to prior works concerning superfast derandomization into cs-$\mathcal{NP}$ protocols. This is since prior works studied simulating subclasses of $prBPP$, rather than simulating all of $\mathcal{MA}$. Specifically, in [CT23b] they deduced free lunch derandomization of doubly efficient interactive proof systems into cs-$\mathcal{NP}$ protocols (and relied on a non-batch-computability and on the classical Nisan-Wigderson generator). And in [CRT25], they simulated uniform-$\mathcal{NC}$ by cs-$\mathcal{NP}$ protocols with a near-linear-time verifier (and relied on hardness for polylogarithmic space).[6]

A refined version: Free lunch derandomization from non-deterministic hardness. Note that in Theorem 1.5, the hard function is computable in $\mathcal{FP}$ yet is hard for smaller $\mathcal{MA}$ time. This was stated only for simplicity (and such an assumption also suffices to derandomize $\mathcal{BPTIME}$ into heur-$\mathcal{DTIME}$). As one might expect, our more refined technical result relies on a non-deterministic upper bound and on a corresponding non-deterministic lower bound.

Loosely speaking, we deduce the conclusion of Theorem 1.5 from the existence of a function computable in cs-$\mathcal{NTIME}[n^{1+O(\epsilon)}]$ that is hard for cs-$\mathcal{MATIME}[n^{1+\epsilon}]$ over all polynomial-time samplable distributions. That is, the upper bound and the lower bound are both non-deterministic, and both require only computational soundness. For formal definitions and a discussion of the assumption, see Sections 2.2 and 6.2, and Theorem 6.7.

The proof of this result relies on an interesting technical contribution. Specifically, we construct a superfast targeted generator that is suitable for the non-deterministic setting, and that has properties useful for working with *protocols that have computational soundness* (e.g., cs-$\mathcal{NP}$ protocols). The construction relies on a near-linear PCP that has an efficient *proof of knowledge*, and specifically on the construction of Ben-Sasson *et al.* [BSCG+13]. See Section 2.2 for details.

1.4 Non-batch-computability from worst-case hardness

As another indication that the non-batch-computability assumption in [CT21a; CT23b] is a red herring (or rather, an ad-hoc feature that can be replaced by various others), we prove that non-batch-computability over the *uniform distribution* follows from standard worst-case hardness assumptions. Indeed, this should not be surprising, since non-batch-computability is reminiscent of hardness of direct product (i.e., it is harder to compute k instances of a function $f(x_1), ..., f(x_k)$ than to compute one instance $f(x)$). However, direct product hardness is known for limited models, whereas we are interested in hardness for general models (see, e.g., [Sha03]).

Loosely speaking, we prove a general result asserting that if a function f is downward self-reducible and efficiently arithmetizable (i.e., admits an efficient low-degree extension), then hardness of f implies non-batch-computability of a related function f' (see Theorem 7.5). The point is that many natural functions have these properties, and thus hardness for any of these functions

[6] The conclusions in both works are incomparable to the conclusion in Theorem 1.5. This is since in [CT23b], the honest prover does not receive a witness in a witness-relation for the problem; whereas in [CRT25] the cs-$\mathcal{NP}$ verifier runs in near-linear time even if the uniform $\mathcal{NC}$ circuit is of large polynomial size. We also note that the main result in [CRT25] simulates a huge class (i.e., $\mathcal{PSPACE}$) by cs-$\mathcal{NP}$ protocols; this is again incomparable to Theorem 1.5, since the latter result focuses on tight time bounds (i.e., free lunch derandomization) whereas the former does not.

implies the existence of non-batch-computable functions. For example, consider the k-Orthogonal Vectors problem (k-OV), in which we are given k sets $A_1, ..., A_k \subseteq \{0,1\}^d$ with $|A_i| = n$, and we want to decide whether there are $a_1 \in A_1, ..., a_k \in A_k$ such that $a_1 \cdot ... \cdot a_k = 0$ (this problem is a cornerstone of fine-grained complexity; see, e.g., [Wil15; Wil18]). Then:

Theorem 1.6 (non-batch-computability from worst-case hardness; informal, see Corollary 7.14)**.** *If k-OV cannot be solved in randomized time $n^{k-o(1)}$, then for some $\epsilon, \delta > 0$ there is a function $f\colon \{0,1\}^n \to \{0,1\}^{r=n^\delta}$ such that:*

1. *Each output bit $f(x)_i$ can be computed in time $T(n) = \tilde{O}(n^k)$.*
2. *For every probabilistic algorithm A running in time $T \cdot r^\epsilon$, with probability 0.99 over $x \in \{0,1\}^n$ it holds that $\Pr_{A, i \in [r]}[A(x) = f(x)_i] < 0.99$.*

The reduction that we show of worst-case computation to approximate-batch-computing is based on a direct product theorem for computing low-degree polynomials, which is targeted specifically for the setting of a k-wise direct product for a small k. See Section 2.3 for details.

2 Technical overview

Free lunch derandomization relies on targeted pseudorandom generators (targeted PRGs), as introduced by Goldreich [Gol11].[7] Targeted PRGs get input x and produce pseudorandomness for uniform algorithms that also get access to the same input x.

In recent years, three types of constructions of targeted PRGs have been developed: Targeted PRGs based on the classical Nisan-Wigderson PRG [NW94] (see, e.g., [CT21a, Section 2.1], or [LP22a; LP22b]); targeted PRGs based on the doubly efficient interactive proof system of Goldwasser, Kalai and Rothblum [GKR15], introduced by Chen and Tell [CT21a] (see, e.g., [CLO+23; CT23b; CTW23; DPT24; LPT24]); and targeted PRGs suitable for non-deterministic derandomization, a-la $\mathcal{MA} \subseteq \mathcal{NP}$, which are based on PCPs and were introduced by van Melkebeek and Sdroievski [MMS23] (see also [MS23]). For a survey see [CT23a].

The point is that none of the constructions above suffice to get free lunch derandomization from our assumptions. Generators based on NW can be very fast, but they rely on assumptions such as non-batch-computability; generators based on interactive proofs are slow, and this obstacle seems inherently difficult to bypass (see Section 2.1); and known PCP-based generators for the non-deterministic setting lack specific features that we need in our setting (see Section 2.2). We will thus have to develop new targeted PRGs, which are fast and rely on standard assumptions, and then use additional ideas to leverage them and obtain our results.

Organization. In Section 2.1 we describe the construction underlying the proof of Theorem 1.2, and in Section 2.2 we describe the construction underlying the proof of Theorem 1.5 and of its technical extension. In Section 2.3 we describe the proof of Theorem 1.6.

[7] Recall that for free lunch derandomization we cannot rely on standard PRGs for non-uniform circuits. First, a PRG for size-T circuits must have seed length at least $\log(T)$, in which case enumerating over seeds (and evaluating a T-time algorithm) yields derandomization in quadratic time. Secondly, such a PRG necessitates circuit lower bounds, and we want constructions from hardness for uniform algorithms.

2.1 A superfast targeted generator, and proof of Theorem 1.2

We first explain why known proof-system-based generators are slow, even when using very fast proof systems, and what is our main idea for doing better. Readers who are only interested in a self-contained technical description of the new generator may skip directly to Section 2.1.2.

2.1.1 Generators based on interactive proofs, and their drawbacks

Recall that the generator of Chen and Tell [CT21a] relies on an interactive proof system (specifically, that of [GKR15], but let us discuss their idea more generally). For each round i of the proof system, they consider the prover strategy P_i on input x as a function of the verifier's challenges up to that point, and use the *truth-table* of P_i as a hard truth-table for a classical PRG (say, [NW94] or [SU05]). The classical PRG yields a list L_i of pseudorandom strings, and they output $\cup_i L_i$.[8]

One would hope that using superfast proof systems, wherein the prover's strategy function is computable in near-linear time (e.g., [CMT12]), would yield a superfast generator. However, this idea faces inherent challenges. First, the generator uses the *truth-table* of P_i; thus, even if the prover's strategy is computable in near-linear time, computing it over all possible verifier challenges (across all i's) would take at least quadratic time. Secondly, the prover's response at round i depends not only on the verifier's challenge in round i, but also on the challenges at *previous* rounds. Thus, we need each P_i to depend on sufficiently few past responses so that the truth-table of P_i is only of super-linear size. Proof systems that we are aware of, even ones with very fast provers, yield generators with large polynomial overheads due to both problems.

Beyond these technical problems, more generally, *interactive proofs do not seem to be the right tool for the job*. To see this, observe that this generator relies on a small sequence of long, static "proofs" (i.e., the P_i's) that are all committed to *in advance*. Indeed, this is far more similar to a PCP than to an interactive proof system. The key to our new construction is using technical tools underlying classical PCP constructions in order to replace the proof system of [GKR15]. Specifically, we will construct a superfast targeted generator using a *highly efficient sorting network*.

2.1.2 A superfast generator based on highly efficient sorting networks

We want a generator that gets input $x \in \{0,1\}^n$, runs in near-linear time in T, and produces pseudorandomness for $T^{.99}$-time algorithms that get x. We will prove the following:

Theorem 2.1 (superfast targeted generator; informal, see Theorem 4.8)**.** *Let $\{C_n\}$ be a sufficiently uniform circuit family of size T and depth d. Then, for every sufficiently small constant $\delta \in (0,1)$ there is a deterministic algorithm SPRG_C and a probabilistic oracle algorithm Rec_C such that:*

1. **Generator.** *When SPRG_C gets input $x \in \{0,1\}^n$ it runs in time $d \cdot T^{1+O(\sqrt{\delta})}$ and outputs a list of T^δ-bit strings.*
2. **Reconstruction.** *Suppose that Rec_C gets input $x \in \{0,1\}^n$ and oracle access to a function $D\colon \{0,1\}^{T^\delta} \to \{0,1\}$ that is a $(1/T^\delta)$-distinguisher for $\mathsf{SPRG}_C(x)$. Then, $\mathsf{Rec}_C(x)$ runs in time $(d+n) \cdot T^{O(\sqrt{\delta})}$, makes $T^{O(\sqrt{\delta})}$ queries to D, and with probability at least $2/3$ outputs $C_n(x)$.*

The main part in proving Theorem 2.1 is an encoding of the computation of $C_n(x)$ as a bootstrapping system, a-la [IW98; CT21a] (see definition below). In order to prove Theorem 2.1

[8] The generator also relies on specific features of the P_i's, namely that they yield a downward self-reducible sequence of codewords, but let us ignore this fact for a moment.

we need an extremely efficient bootstrapping system, which is significantly faster than previously known constructions [CT21a; CTW23; CLO+23; DPT24; LPT24].

Standard setup. From here on, let $\epsilon = \Theta(\delta)$ be sufficiently small. The generator computes $C_n(x)$ and then encodes the gate-values obtained in $C_n(x)$ during the computation as a bootstrapping system, which is a sequence of functions $\{P_1, ..., P_{d'}\}$ ("layers") with the following properties:

1. **Error-correction:** Each layer $P_i \colon \mathbb{F}^m \to \mathbb{F}$ is a low-degree polynomial.
2. **Base case:** Computing each entry of P_1 can be done in time $t = T^\epsilon$, given x.
3. **Downward self-reducibility (DSR):** For $i > 1$, computing each entry of P_i reduces in time t to computing entries in P_{i-1}.
4. **Faithful representation:** Computing each entry of $f(x)$ can be done in time t given P_d.

It will be crucial for us that $d' \approx d$, that each P_i is a function over a domain of size $|\mathbb{F}|^m \approx T$, and that the generator can compute the bootstrapping system from x in near-linear time in T.

Warm-up: A nicely arranged grid. Observe that the gate-values of $C_n(x)$ are already arranged into d layers $p_1, ..., p_d \colon [T] \to \{0,1\}$ with built-in DSR.[9] For convenience, let us replicate each gate to "left" and "right" copies; that is, for every $i \in [d]$ there are now $2T$ gates in layer p_i, indexed by $(g, b) \in [T] \times \{\mathsf{lt}, \mathsf{rt}\}$ such that $p_i(g, \mathsf{lt}) = p_i(g, \mathsf{rt})$. Also, let us arithmetize the input layer p_1 in the standard way: Using a set $H \subseteq \mathbb{F}$ of size $|h| = T^\epsilon$ and m such that $|H|^m \geq 2T$, we define $P_1 \colon \mathbb{F}^m \to \mathbb{F}$ to be a low-degree polynomial (i.e., of individual degree $h - 1 \approx T^\epsilon$) such that if $\vec{w_{j,b}}$ is the $(j,b)^{th}$ element of H^m, then $P_1(\vec{w_{j,b}}) = p_1(j, b)$. Since P_1 is, essentially, just a low-degree extension of $x \in \{0,1\}^n$, it is computable at any input in time $\tilde{O}(n) \leq \tilde{O}(T)$.

Now, imagine for a moment that the circuit is a $d \times 2T$ grid such that every gate takes its inputs from the pair of gates *directly* below it. Formally, for every $(g, b) \in [T] \times \{\mathsf{lt}, \mathsf{rt}\}$, the value of $p_{i+1}(g, b)$ depends on $p_i(g, \mathsf{lt})$ and $p_i(g, \mathsf{rt})$. Then, constructing a bootstrapping system is much easier. Specifically, starting from $i = 1$ and working our way up, we have a low-degree polynomial P_i for p_i, and we can easily construct a polynomial P_{i+1} for p_{i+1}:

$$P_{i+1}(\vec{w_{j,b}}) = B_{i+1}\big(P_i(\vec{w_{j,\mathsf{lt}}}), P_i(\vec{w_{j,\mathsf{rt}}})\big) \ ,$$

where B_{i+1} is a low-degre arithmetization of the Boolean gate operation in layer $i + 1$. Note that P_{i+1} defined above is a low-degree polynomial that agrees with p_{i+1} on $H^m \equiv [2T]$, and that the resulting sequence $P_1, ..., P_d$ is downward self-reducible in the straightforward way.

Even in this easy setting we are not done, since when naively propagating this construction up across layers, the degree blows up (because $\deg(P_{i+1}) = c \cdot \deg(P_i)$ for some constant $c > 1$ that depends on the gate operation). However, we can maintain individual degree at most $h - 1$, using the linearization idea of Shen [She92]. Specifically, after each P_i (and before P_{i+1}), we will add a sequence of polynomials $P_{i,j}$ that decrease the individual degree of each variable at a time to $h - 1$, while maintaining the behavior of P_i on H^m (see Section 4.1.1 for an implementation). Of course, after adding this sequence, it is no longer guaranteed that each gate in p_{i+1} takes its inputs directly from the two gates below it in $P_{i,\ell}$ (where $P_{i,\ell}$ is the last polynomial in the degree-reduction sequence). But observe that *if this property would hold, we would be done.*

[9] Indeed, we will assume that the circuit is sufficiently uniform, and in particular that given $(i, j) \in [d] \times [T]$ we can output the indices of gates in layer $i - 1$ that feed into gate j at layer i, in time $\ll T$.

The key component: Simple sorting. The crux of our proof is *sorting* the gate-values at each layer such that after sorting, the value of each gate g will be directly below the gates to which g feeds in the layer above. Towards this operation, we assume that the circuit is sufficiently uniform, in the following sense: Each gate has fan-out two, and there is a uniform formula of size polylog(T) that gets input $(i, g, \sigma) \in [d] \times [T] \times \{0,1\}$ and prints the index of the σ^{th} output gate of g in layer $i+1$ (see Definition 4.1).[10] In particular, we can arithmetize this formula and obtain low-degree polynomials $OUT_{i,b}$ computing the index of the b^{th} output gate (i.e., left or right) of g. Then, for each P_i, we obtain a low-degree P'_i that maps (g, b) to $(OUT_{i,b}(g), P_i(g, b))$.

We now sort the values of P'_i such that the value that originally appeared in location (g, b) (i.e., $P_i(g, b)$) will appear in location $(OUT_{i,b}(g), b)$ after sorting, for some $b \in \{\mathsf{lt}, \mathsf{rt}\}$. That is, we construct a sequence of polynomials $P'_{i,1}, ..., P'_{i,\ell}$ such that, thinking of g' as an index of a gate in layer $i+1$, we have that $\left\{P'_{i,\ell}(g', b)\right\}_{b\in\{\mathsf{lt},\mathsf{rt}\}} = \{P'_i(IN(g', b))\}_{b\in\{\mathsf{lt},\mathsf{rt}\}}$, where $IN(g', b)$ is index index of the b^{th} gate feeding into g'. We do this by arithmetizing the operations of a sorting procedure that runs in parallel time $\ell = \text{polylog}(T)$ (and thus we only add ℓ polynomials).

We need a sorting procedure that is arithmetizable as a sequence of polynomials that are simultaneously of low-degree and downward self-reducible; that is, each $P_{i,j}$ is of low degree, and the value of $P_{i,j}(g, b)$ depends in a simple way on a small number of locations in $P_{i,j-1}$ whose indices are easily computable from (g, b). This seems like a chicken-and-egg problem, since our goal in constructing a bootstrapping system to begin with is precisely to encode the computation of $C_n(x)$ into a sequence of polynomials that achieve both properties simultaneously. However, we have now reduced this problem to achieving these properties only for the specific computation of a *sorting procedure*, and we can choose which sorting procedure to work with.

The key is using a highly efficient sorting network *whose operations are as simple as possible*. Indeed, recall that sorting networks work in parallel, perform simple operations, and their circuit-structure function is rigid and simple. For our purposes, the most useful network turns out to be Batcher's [Bat68] classical bitonic sorter: The non-recursive implementation of this sorting network uses functionality that is as simple as one might imagine; see Figure 1.[11]

There are still some details left to sort out. We need to arithmetize the operations of the bitonic sorter (see Section 4.1.2), to arithmetize the gate operations (see Section 4.1.3), and to add degree-reduction polynomials in between all operations. For full details, see Section 4.1.

From bootstrapping system to targeted generator. Lifting the bootstrapping system to a proof of Theorem 2.1 is standard by now (see, e.g., [CT21a] for details). In a gist, the generator maps each layer in the bootstrapping system to a list of strings, using the NW generator; and the reconstruction uses a distinguisher to iteratively compress each layer, starting from the input layer and going up until reaching the top layer (which has the output $C_n(x)$). To compress a layer, it uses the reconstruction procedure of NW, which works in small time $T^{O(\sqrt{\delta})}$ when the output length T^{δ} of NW is small (as it will be in our setting; see below).

Note that the overall reconstruction uses $d \ll T$ steps, each running in time $T^{O(\sqrt{\delta})}$ and using access to a T-time distinguisher. If C_n is computable in time $\bar{T} = T^{1+O(\sqrt{\delta})}$ yet hard for time $\bar{T}^{1-\epsilon}$, we obtain a contradiction. See Section 4.2 for details.

[10] Note that many natural functions satisfy this notion of uniformity, since the adjacency relation in circuits for these functions is typically very rigid (and we can add $\log(T)$ layers above each layer to ensure that gates have fan-out two).

[11] In particular, it will be very convenient for our arithmetization that the bitonic sorter compares the values of gate-values whose indices are of the form $\vec{x}$ and $\vec{y} = \vec{x} + \vec{e_i}$ where $\vec{e_i}$ has Hamming weight 1.

2.1.3 Proof of Theorem 1.2

Let us start by derandomizing $\mathcal{RTIME}[T]$. Fix a machine M running in time T and solving a problem with one-sided error. If we instantiate Theorem 2.1 with a function hard over all polynomial-time samplable distribution, when x is chosen from a polynomial-time samplable distribution, will all but negligible probability there exists $s \in \mathsf{SPRG}_C(x)$ such that $M(x,s) = 1$.

Unfortunately, this only yields *quadratic time* derandomization. Specifically, if OWFs exist we can assume wlog that M uses only T^ϵ random coins (since the OWF yields a PRG with polynomial stretch running in near-linear-time; see Assumption 3.7). We instantiate Theorem 2.1 with a hard function computable by circuits of size $T^{1+O(\epsilon)}$ and depth $T^{O(\epsilon)}$, in which case $\mathsf{SPRG}_C(x)$ yields $T^{1+O(\epsilon)}$ pseudorandom strings for $M(x,\cdot)$.[12] However, evaluating $M(x,\cdot)$ on each of those strings (to search for s such that $M(x,s) = 1$) takes time $T^{2+O(\epsilon)}$.

Nevertheless, using Theorem 2.1 we exponentially reduced the number of random coins used by M, from T^ϵ to $(1+O(\epsilon)) \cdot \log(T)$ (since it now suffices to choose from the list $\mathsf{SPRG}_C(x)$), and crucially, we did so *without meaningfully increasing the running time of* M.

Free lunch derandomization with small advice. We now use a stronger property of SPRG_C. Specifically, observe that the generator computes a small number $d' = d \cdot \mathrm{polylog}(T)$ of lists, and for every x such that the reconstruction fails, at least one of the lists is *pseudorandom* for $M(x,\cdot)$. In particular, on inputs x such that $\Pr_r[M(x,r)] \geq 1/2$ we have that $\Pr_{s \in \mathsf{SPRG}_C(x)}[M(x,s) = 1] \geq 1/O(d')$.[13] Our first step is to increase the density of "good" strings s in the list from $1/O(d')$ to $1 - n^{-\omega(1)}$. Naively re-sampling from the list can achieve this while increasing the number of random coins to $O(\log T)^2$, and using randomness-efficient samplers, we do this with $\tilde{O}(\log T)$ random coins, and with no significant increase to the running time of M.

The crucial observation is that now there exists one seed $s \in \{0,1\}^{\tilde{O}(\log T)}$ that is good for *all efficiently samplable distributions*, since there are only countably many such distributions. That is, for every fixed polynomial-time machine sampling a distribution $\mathbf{x} = \{\mathbf{x}_n\}$, note that the probability over $x \sim \mathbf{x}_n$ and $s \in \{0,1\}^{\tilde{O}(\log n)}$ that $\Pr_r[M(x,r) \geq 1/2]$ yet $M(x,s) = 0$ is negligible. By a union-bound, the probability that this holds for at least one of the first (say) n Turing machines is also negligible. Thus, if we fix a good seed $s_n \in \{0,1\}^{\tilde{O}(\log n)}$ as advice to the derandomization algorithm, then for every efficiently samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}$ and every sufficiently large $n \in \mathbb{N}$, with all but negligible probability over $x \sim \mathbf{x}_n$ the derandomization algorithm $M(x, \mathsf{SPRG}_C(x)_{s_n})$ outputs the correct decision in time $T^{1+O(\epsilon)}$.

Loose ends. The argument above only derandomizes $\mathcal{RP}$. However, since it uses a targeted PRG, it also works for promise-problems; in particular, the argument shows that $pr\mathcal{RTIME}[T] \subseteq \mathsf{heur}\text{-}\mathcal{DTIME}[T^{1+O(\epsilon)}]$. Now we can imitate the standard non-black-box reduction of derandomization of $pr\mathcal{BPP}$ to derandomization of $pr\mathcal{RP}$ (as in [BF99]), while noting that all the inputs to $pr\mathcal{RTIME}$ considered in this reduction are explicitly computable in polynomial time. Thus, if this reduction fails with noticeable probability over $x \sim \mathbf{x}_n$, then an efficient adversary can find

[12] Here is a sketch of the standard analysis. The hard function is computable in time $\bar{T} = T^{1+O(\epsilon)}$, but hard for time $\bar{T}^{1-\epsilon}$. Note that the reconstruction runs in time $T^{O(\epsilon)}$ and makes at most $T^{O(\epsilon)}$ to its distinguisher D. We will use $D_x = M(x,\cdot)$, which runs in time T. In this case the reconstruction runs in time $T^{1+O(\epsilon)} = \bar{T}^{1-\epsilon}$. If there is a polynomial-time samplable distribution such that with noticeable probability the derandomization fails, then there is a polynomial-time samplable distribution such that with noticeable probability, the reconstruction computes the hard function too quickly. See Theorem 5.1 for further details.

[13] That is, the generator is actually a somewhere-PRG; a similar property of the generator of [CT21a] was used in the past, see [CRT22; DPT24], albeit for different purposes.

an input on which the derandomization of $prRTIME$ fails, which is a contradiction. Moreover, the compositional overhead caused by this reduction is small when we focus on derandomization in near-linear time $T^{1+O(1)}$. See Theorem 3.5 for details.

Lastly, as mentioned in Section 1.2.2, we can eliminate the $\ell = \tilde{O}(\log n)$ bits of advice, by assuming a PRG that stretches ℓ^ϵ bits to ℓ bits in time $T^{1+O(\epsilon)}$ and fools uniform machines running in slightly smaller time. See the proof of Theorem 5.1 for details.

2.2 A superfast targeted generator based on PCPs with proofs of knowledge

As a warm-up, let us first prove Theorem 1.5. Recall that we have a function f computable in near-linear time $n^{1+O(\epsilon)}$ that is hard for smaller $\mathcal{MA}$ time $n^{1+\epsilon}$ over all polynomial-time samplable distributions, and we want to simulate $\mathcal{MATIME}[T]$ in cs-$\mathcal{NTIME}[T^{1+O(\epsilon)}]$.

A bare-bones version of [MS23]. We use a variant of the targeted generator of van Melkebeek and Sdroievski [MS23]. Fix $L_0 \in \mathcal{MATIME}[T]$, decided by a verifier V_0. The generator is given $x \in \{0,1\}^n$, and it guesses a witness w for V_0. It also guesses $f(x,w)$, and a PCP witness π for the language $L = \{((x,w), f(x,w))\}$, which is decidable in time $|(x,w)|^{1+O(\epsilon)} = T(|x|)^{1+O(\epsilon)}$. The generator then verifies that π is indeed a convincing witness (by enumerating the $(1+\epsilon)\cdot\log(T)$ coins of the PCP verifier), and outputs the NW PRG with π as the hard truth-table.

Our deterministic verifier V for L_0 gets x and a witness $\bar{w} = (w, f(x,w), \pi)$, uses this generator with $(x, w, f(x,w), \pi)$, and outputs $\wedge_{s\in \mathsf{NW}(\pi)} V_0(x,w,s)$.[14] This verifier indeed runs in time $T^{1+O(\epsilon)}$, assuming that we use a fast PCP and relying on the OWF assumption.[15] Now, assume that an efficient dishonest prover $\tilde{P}$ can find, with noticeable probability, an input $x \notin L_0$ and a proof $\bar{w} = (w, f(x,w), \pi)$ such that $V(x,\bar{w}) = 1$. We show that on the fixed input (x,w), an $\mathcal{MA}$ reconstruction procedure succeeds in computing $f(x,w)$ in time $T^{1+\epsilon}$. We stress that f is only hard over polynomial-time samplable distributions, and thus we can only use this argument to deduce that no *efficient adversary* can find x and $\bar{w}$ that mislead the verifier; that is, we derandomize $\mathcal{MATIME}[T]$ into cs-$\mathcal{NTIME}[T^{1+O(\epsilon)}]$.

How does the $\mathcal{MA}$ reconstruction work for such a fixed (x,w)? By the above, there is π such that $D_{x,w}(r) = V_0(x,w,r)$ is a distinguisher for $\mathsf{NW}(\pi)$. Given (x,w) as input, we run the reconstruction algorithm R_{NW} of NW. Recall that when R_{NW} gets suitable advice (which consists of random coins, and bits from the "correct" π), it uses $D_{x,w}$ to build a concise version of π. We non-deterministically guess advice for R_{NW}, and this determines a concise version of a PCP witness π'. We then guess $f(x,w)$ and run the PCP verifier on input $((x,w), f(x,w))$, giving it oracle access to π'. The point is that: (1) The running time of this entire procedure is small, and can be made $T^{1+\epsilon}$; (2) There is a guess such that this procedure accepts with probability 1, due to the perfect completeness of the PCP verifier; (3) After guessing the advice for R_{NW}, it commits to a single PCP witness π', and thus if we guessed $f(x,w)$ incorrectly the PCP verifier will reject, with high probability. For precise details see the proof of Theorem 6.4.

A non-deterministic hardness assumption, and a superfast generator with witnessable soundness. Next, we would like to relax the hardness assumption, and only require that f will be

[14] Throughout the paper we assume that $\mathcal{MA}$ verifiers have perfect completeness.

[15] Specifically, we use a PCP in which PCP proofs for $\mathcal{DTIME}[T]$ can be computed in time $T^{1+o(1)}$. Also, relying on the OWF assumption, we can assume wlog that the $\mathcal{MA}$ verifier only uses T^ϵ random coins. Indeed, when the $\mathcal{MA}$ verifier uses T^ϵ random coins, we can instantiate NW with small output length $T^\epsilon \approx |\pi|^\epsilon$, in which case it runs in time $|\pi|^{1+O(\epsilon)} = T^{1+O(\epsilon)}$ and produces a list of size $T^{O(\epsilon)}$.

computable non-deterministically. Since are constructing a cs-$\mathcal{NP}$ protocol for $L \in \mathcal{MA}$, we need an efficient honest prover for f, and we thus use a hard function $f \in \text{cs-}\mathcal{NTIME}[n^{1+O(\epsilon)}]$.[16]

The cs-$\mathcal{NP}$ verifier for L is similar to the one in the "bare-bones" version above. The only difference is that now it also gets a witness w_f for f, uses a cs-$\mathcal{NP}$-verifier V_f to compute $V_f((x,w),w_f)$ (which hopefully outputs $f(x,w)$), and applies NW to a PCP proof π_{x,w,w_f} for the $T^{1+O(\epsilon)}$-time computable language $\{((x,w,w_f),V_f((x,w),w_f))\}$.

The main challenge is that with this new generator, the reconstruction procedure described above is not an $\mathcal{MA}$ protocol anymore. To see why, consider an efficient adversary $\tilde{P}$ that finds x and (w,w_f,π) such that $D_{x,w}(r)$ is a distinguisher for $\mathsf{NW}(\pi_{x,w,w_f})$. The reconstruction procedure then quickly and non-deterministically computes $V_f((x,w),w_f)$. The key problem is that V_f may err in computing f on some hard-to-find inputs. Specifically, on some inputs (x,w), the reconstruction procedure has several possible outputs: It may be that $D_{x,w}$ is a distinguisher for $\mathsf{NW}(\pi_{x,w,w_f})$ such that $V_f((x,w),w_f) = f(x,w)$, and also for $\mathsf{NW}(\pi_{x,w,w'_f})$ such that $V_f((x,w),w'_f) \neq f(x,w)$; in this case, on different non-deterministic guesses the reconstruction procedure may output either $V_f((x,w),w_f)$ or $V_f((x,w),w'_f)$.

The issue above seems inherent to the common techniques in hardness-vs-randomness (see Remark 6.10 for an explanation). Thus, as explained in Section 1.3, we will rely on a hardness assumption for stronger $\mathcal{MA}$-type reconstruction procedures, in which misleading input-proof pairs do exist but are infeasible to find (i.e., for cs-$\mathcal{MA}$ protocols; see below).

It is still unclear why the protocol above should have such computationally soundness – after all, maybe an adversary can indeed find a witness for the reconstruction (i.e., non-deterministic guesseses for the protocol, and in particular a PCP witness π_{x,w,w'_f}) that cause the reconstruction to output an incorrect value (i.e., output $V_f((x,w),w'_f) \neq f(x,w)$). To ensure that no efficient adversary can do this, we construct the following superfast targeted generator, which has additional properties. The primary one is "witnessable soundness": A witnessing algorithm can efficiently map convincing witnesses for the reconstruction procedure into convincing witnesses for V_f. Thus, since misleading input-witness pairs for V_f are infeasible to find, misleading input-witness pairs for the reconstruction of the new generator are also infeasible to find.

Theorem 2.2 (superfast targeted PRG whose reconstruction has witnessable soundness; informal, see Theorem 6.8)**.** *Consider V_f that gets (z,w') of length $N^{1+O(\epsilon)}$ and runs in linear time $N^{1+O(\epsilon)}$. For every $\delta > 0$ there is a deterministic NGen_V and a probabilistic NRec_V that satisfy the following:*

1. **Generator.** *When NGen_V gets (z,w') it runs in time $N^{1+O(\epsilon+\sqrt{\delta})}$, and if $V_f(z,w')$ does not reject, it prints a list of N^δ-bit strings (otherwise, it outputs $\perp$).*

2. **Reconstruction.** *The reconstruction NRec_{V_f} gets input z and oracle access to $D\colon \{0,1\}^{N^\delta} \to \{0,1\}$, runs in time $N^{1+\epsilon}$, guesses a witness π', tosses random coins r, makes at most $N^{O(\sqrt{\delta})}$ oracle queries, and satisfies the following.*

   (a) (**Efficient honest prover.**) *There is a ppt oracle machine Prv_{V_f} such that for every (z,w') such that $\mathsf{NGen}_{V_f}(z,w') \neq \perp$, and every $(1/M)$-distinguisher D for $\mathsf{NGen}_{V_f}(z,w')$, the probability that $\mathsf{Prv}^D_{V_f}$ convinces NRec^D to output $V_f(z,w')$ is at least $2/3$.*[17]

[16] As in any argument system, the notion of cs-$\mathcal{NP}$ for $L \in \mathcal{MA}$ is non-trivial only when the honest prover is efficient (at least when given a witness in an arbitrary $\mathcal{MA}$-relation for L).

[17] That is, with probability at least $2/3$, the output π' of $\mathsf{Prv}^D_{V_f}(z,w')$ satisfies $\Pr_r[\mathsf{NRec}^D_{V_f}(z,r,\pi') = V_f(z,w')] = 1$.

(b) (**Witnessable soundness.**) *There is a ppt oracle machine* Wit_{V_f} *satisfying the following. For any* D *and any* π'*, if* $\Pr_r[\mathsf{NRec}^D_{V_f}(z,r,\pi') = y] \geq 1/2$ *for some* $y \in \{0,1\}^*$*, then with probability at least* $2/3$ *the algorithm* $\mathsf{Wit}_{V_f}(z,\pi')$ *outputs* w' *such that* $V_f(z,w') = y$*.*

The construction of Theorem 2.2 is based on PCPs with proofs of knowledge, as defined by Ben-Sasson *et al.* [BSCG+13]. This notion asserts that convincing PCP witnesses can be efficiently "translated back" to satisfying assignments for the original predicate that the PCP proves. We will use the specific construction from [BSCG+13], since we crucially need a PCP that is both very fast (i.e., has short witnesses that can be computed by a near-linear-time prover) and that has a polynomial-time proof of knowledge. The details appear in Theorem 6.8.

The derandomization result itself is stated in Theorem 6.7. The specific assumption that it relies on is function $f \in \text{cs-}\mathcal{NTIME}[n^{1+O(\epsilon)}, \text{poly}(n)]$ that is hard for $\text{cs-}\mathcal{MATIME}[n^{1+\epsilon}, n^2]$ protocols over all polynomial-time samplable distributions, where the second quantitative term in both expressions denotes the runtime of the honest prover in the protocol.[18] That is, for every $\text{cs-}\mathcal{MATIME}[n^{1+\epsilon}, n^2]$ protocol with a verifier V and an honest prover P such that the protocol is computationally sound, it is infeasible to find inputs z on which P manages to convince V of the correct value of $f(z)$. We make this notion quantitatively precise in Definition 6.6 and in Theorem 6.7. The rationale behind the hardness assumption is the obvious one: If we disregard randomness, the verifier in the upper bound has more power than the verifier in the lower bound. As an additional sanity check, a random function with (say) n^ϵ output bits also attains this hardness; and indeed, it is even plausible that there is a function in $\mathcal{FP}$ (rather than only $\text{cs-}\mathcal{NP}$) with such hardness.

2.3 Worst-case to approximate-batch-case reductions for natural functions

Informally, a function $g : \{0,1\}^n \to \{0,1\}^k$ is non-batch computable if any individual output bit $g(x)_i$ can be computed in time T, but no algorithm running in time significantly faster that $T \cdot k$ can correctly print all of the k output bits $g(x)$ (or a large fraction of them).

We prove that non-batch-computability assumption follows from any *worst-case* hard decision problem f that admits two natural properties:

1. An efficient low-degree extension: There is a low-degree multivariate polynomial p that computes f on binary-valued inputs ($\forall x \in \{0,1\}^n, f(x) = 0 \iff p(x) = 0$) and is computable in time proportional to f. We denote $d = \deg(p)$.

2. Downward-self-reducibility: Solving a single instance of the problem efficiently reduces to solving many smaller instances.

Given such a hard function, we show that the k-wise direct product of p is non-batch-computable; that is, no algorithm can print a large fraction of the outputs significantly faster than computing each output independently. The key lemma is a direct product theorem that is akin to the ones in Ball *et al.* [BRS+18; BGH+24], but focuses on the case in which the number of instances $k = n^\gamma$ is small. (In contrast, in prior work [BRS+18; BGH+24] the number of instances $k = \text{poly}(n)$ was significantly larger the size n of each instance.)

[18] The reason for bounding the running times of the honest provers is that we are computing a function f that does not necessary have an underlying witness-relation. Again, this is standard when considering argument systems, and $\text{cs-}\mathcal{NP}$ systems were also defined this way in prior work (see, e.g., [Gol01, Section 4.8] and [CT23b; CRT25]). We note that in our assumption we will allow the honest prover in the $\text{cs-}\mathcal{MA}$ protocol to get a witness in an underlying witness-relation that can be efficiently generated; see Theorem 6.7 for precise details.

The main idea. We show that any batch-solver fails on a small constant fraction $\delta > 0$ of the k-tuples, and later explain how to lift this to hardness over $1 - \delta$ of the k-tuples.

Let us first attempt a naive reduction and see where it fails. Assume towards a contradiction that a batch-solver B succeeds on more than $1 - \delta$ of the k-tuples. Given a ("large") instance X to the problem, we apply downward self-reduction to obtain k' smaller instances $x_1, ..., x_{k'}$; a correct solution to all k' smaller instances can be easily combined into a solution for X. Since the batch-solver only succeeds on average and we need to solve all k' instances correctly, a natural idea is to use error-correction: We randomly sample a low-degree curve C passing through $x_1, ..., x_{k'}$, apply the batch-solver on a set of $k = O(d \cdot k')$ points on this curve, and uniquely decode from $\approx \delta$ errors to obtain the unique degree-$(d \cdot k')$ polynomial $p \circ C$ that agrees with $B \circ C$.

The only problem with this idea is that the points on the curve on which we apply the batch-solver B are not uniformly distributed, and hence B is not guaranteed to succeed with probability $1 - \delta$. The main idea to resolve this is to add additional error-correction. Specifically, assume that $x_1, ..., x_{k'}$ are embedded in points $C(1), ..., C(k')$ on the curve, and that we decode using the points $C(k' + 1), ..., C(k' + k)$ on the curve. For each $i \in \{k' + 1, ..., k' + k\}$, we sample a random line L_i passing through $C(i)$ (i.e., $L_i(0) = C(i)$). Now, for each fixed $j \in [O(d)]$, consider the k-tuple that passes through the j^{th} point in each of the k lines $L_1, ..., L_k$

$$\bar{x}_j = (L_1(j), ..., L_k(j)) \ .$$

Observe that for each $j \in [d]$ the k-tuple $\bar{x}_j$ is uniformly random (over choice of C and of L_i's), so we can apply the batch-solver B to it. In particular, by an averaging argument, with high probability over choices of C and L_i's, for most of the points $C(i)$ (where $i \in \{k' + 1, ..., k' + k\}$), for most of the points $j \in [d]$ we have $B(\bar{x}_j)_i = p(L_i(j))$ (i.e., $B(\bar{x}_j)$ is correct on its i^{th} coordinate).[19] Thus, we now apply the Reed-Solomon unique decoder twice:

1. First, for every $i \in \{k' + 1, ..., k\}$ we run the batch-solver B on $\bar{x}_1, ..., \bar{x}_{O(d)}$ and look at its i^{th} coordinate. This yields a sequence of $O(d)$ values, and we uniquely decode the results, hoping to obtain the degree-d polynomial $p \circ L_i$ and evaluate it on 0 to get $p(C(i))$.

2. Secondly, analogously to the naive attempt, we now uniquely decode the sequence of k values obtained in the first step, hoping to obtain the degree-$(d \cdot k')$ polynomial $p \circ C$.

If indeed for most $C(i)$'s it holds that for most $j \in [d]$ we have $B(\bar{x}_j)_i = p(L_i(j))$, then for most $C(i)$'s we correctly obtain the value $p(C(i))$ in the first step, in which case the unique decoder outputs $p \circ C$ in the second step. Then we can compute $p \circ C(1), ..., p \circ C(k')$ and combine the results $p(x_1), ..., p(x_{k'})$ to compute the hard function at input X.

Remaining gaps. There are two remaining gaps in the proof above. First, the proof shows that every batch-solver fails on a small fraction $\delta > 0$ of the inputs, whereas we are interested in showing that every batch-solver fails on a very large fraction $1 - \delta$ of the inputs.

We bridge this gap by applying direct-product *again*, which increases the average-case hardness from δ to $1 - \delta$, carefully adapting well-known techniques from a sequence of works by Impagliazzo *et al.* [IJK07; IJK+10]. Their techniques require a way to verify that a batch-computation is correct,[20] and the key observation is that in our setting, we can efficiently test whether a batch-

[19] Our actual argument will use Chebyshev's inequality (rather than an averaging argument), and to get pairwise independence we will use L_i's that are quadratic curves. For simplicity, we hide this in the high-level overview.

[20] In other words, they only yield a list-decoder rather than a decoder, and we need to weed the list to find which candidate is correct.

solver correctly computes $1-\epsilon$ of the bits of $f(X)$, by sampling $O(1/\epsilon)$ output bits and computing f at each of these bits. (Observe that this does not significantly increase the running time of the batch-solver.)

The second gap is that the proof shows hardness of batch-computing a polynomial over a large field, rather than a Boolean function; we bridge this gap via the standard approach of applying the Hadamard code to the polynomial. For precise details, see Section 7.

3 Preliminaries

The machine model throughout the paper is the RAM model. We will use the notation $\circ$ to denote the concatenation of strings, and also of lists. We recall that standard definition of a distinguisher for a distribution:

Definition 3.1 (distinguisher)**.** *We say that $D\colon \{0,1\}^M \to \{0,1\}$ is an ϵ-distinguisher for a distribution $\mathbf{w}$ over $\{0,1\}^M$ if $\left|\Pr_{r\in\{0,1\}^M}[D(r)=1]-\Pr_{w\sim\mathbf{w}}[D(w)=1]\right|\geq\epsilon$.*

3.1 Derandomization and hardness over samplable distributions

Let us formally define the notion of derandomization over all polynomial-time samplable distributions, where the derandomization refers to promise-problems. We stress that this is a stronger notion (compared to only derandomizing languages), and it is useful (see, e.g., Theorem 3.5). All of our results achieve this stronger notion, since they are based on targeted generators.

Definition 3.2 (simulation over all polynomial-time samplable distributions)**.** *For a promise-problem $\Pi=(\Pi^{\mathsf{Y}},\Pi^{\mathsf{N}})$ and a class prC of promise-problems, we say that $\Pi\in$ heur-prC if there is $\Pi'\in prC$ such that for every probabilistic polynomial-time algorithm F it holds that*

$$\Pr_{x\leftarrow F(1^n)}\left[x\in(\Pi^{\mathsf{Y}}\setminus(\Pi')^{\mathsf{Y}})\cap\{0,1\}^n\vee x\in(\Pi^{\mathsf{N}}\setminus(\Pi')^{\mathsf{N}})\cap\{0,1\}^n\right]\leq n^{\omega(1)}\,.$$

We now recall the definition of hardness over all polynomial-time samplable distributions, and prove that such hardness is necessary for derandomization over all polynomial-time samplable distributions.

Definition 3.3 (hardness over all polytime samplable distributions)**.** *We say that $f\colon\{0,1\}^*\to\{0,1\}^*$ is hard for probabilistic time T over all polynomial-time samplable distributions if the following holds. For every probabilistic time-T algorithm A, and every polynomial-time samplable distribution $\mathbf{x}=\{\mathbf{x}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n\in\mathbb{N}$, we have $\Pr_{x\sim\mathbf{x}_n}[A(x)=f(x)]\leq\mathsf{negl}(n)$.*

Claim 3.4 (hardness over all polytime samplable distributions is necessary for free lunch derandomization)**.** *Let $T(n)$ be a polynomial, let $\epsilon>0$, and assume that $prBPTIME[T]\subseteq$ heur-$prDTIME[T^{1+\epsilon}]$. Then, there is a function $f\colon\{0,1\}^n\to\{0,1\}^{\log(n)}$ computable in deterministic time $T^{1+3\epsilon}$ such that for every probabilistic time-T machine M and every polynomial-time samplable distribution $\mathbf{x}=\{\mathbf{x}_n\}_{n\in\mathbb{N}}$, with all but negligible probability over $x\sim\mathbf{x}_n$ it holds that $\Pr[M(x)=f(x)]<2/3$.*

Proof. Recall that there is a function $f\colon\{0,1\}^n\to\{0,1\}^{\log(n)}$ that is computable in deterministic time $T^{1+3\epsilon}$ and yet is hard for deterministic time $T^{1+2\epsilon}$ on all but finitely many inputs (see, e.g., the proof of [CT21a, Claim 5.2]).

Assume towards a contradiction that there is a T-time machine M and a polynomial-time samplable distribution $\mathbf{x}$ such that with noticeable probability $1/p(n)$ over $x \sim \mathbf{x}_n$ it holds that $\Pr[M(x) = f(x)] \geq 2/3$. Let $\Pi = (\Pi^{\mathsf{Y}}, \Pi^{\mathsf{N}})$ where

$$\Pi^{\mathsf{Y}} = \{(x,i,b) : \Pr[M(x)_i = b] \geq 2/3\} \quad \text{and } \Pi^{\mathsf{N}} = \{(x,i,b) : \Pr[M(x)_i = b] \leq 1/3\} \ ,$$

and note that $\Pi \in prBPTIME[T]$, and that with probability at least $1/p(n)$ over $x \sim \mathbf{x}_n$, for all $i \in [\log(n)], b \in \{0,1\}$ it holds that $(x,i,b) \in \Pi$ and $(x,i,b) \in \Pi^{\mathsf{Y}} \iff f(x)_i = b$.

By our assumption, $\Pi \in \text{heur-}prDTIME[T^{1+\epsilon}]$. Let M_L be a deterministic $T^{1+\epsilon}$-time machine for L. Consider a machine A that gets input x, runs M_L on all $i \in [\log(n)], b \in \{0,1\}$, and if M_L yields a consistent output (i.e., for every i there is exactly one b such that $M_L(x,i,b) = 1$) then $A(x)$ prints this output. By the above, there is a $1/p(n)$ fraction of inputs such that $A(x) = f(x)$. However, A runs in time less than $T^{1+2\epsilon}$, a contradiction. ■

3.1.1 Two-sided error vs one-sided error

The following result asserts that derandomization of randomized algorithms with one-sided error implies derandomization of randomized algorithms with two-sided error, even when the context is only derandomization over all polynomial-time samplable distributions. Indeed, the proofs imitates the standard reduction [BF99], while observing that all inputs generated in the reduction are explicit (i.e., generated in polynomial time), and hence the derandomization should work on all of them (with all but negligible probability).

Theorem 3.5. *Let $\epsilon \in (0,1)$ be sufficiently small. Assuming OWFs exist, and suppose that for every polynomial T, there is a targeted hitting-set generator that gets input $x \in \{0,1\}^n$, runs in time $T^{1+\epsilon}$, outputs a list of at most T^{ϵ} strings of length T, and fools all T-time algorithms with one sided-error, over all polynomial-time samplable distributions.*[21] *Then,*

$$prBPTIME[T] \subseteq \text{heur-}prDTIME[T^{1+O(\epsilon)}] \ .$$

Moreover, if the targeted hitting set needs $\alpha = \alpha(T)$ bits of advice, the final heuristic derandomization needs $\alpha' = 2 \cdot \alpha(T^{1+O(\epsilon)})$ bits of advice.

Proof. We first consider the case where the targeted hitting set generator requires no advice, and then explain how to handle the case with advice similarly.

Let A be a T-time algorithm deciding a promise problem $\Pi = (\Pi^{\mathsf{Y}}, \Pi^{\mathsf{N}}) \in prBPTIME[T]$. Applying the assumed OWFs and a subsequent error reduction, we masy assume that for an arbitrary small constant $\epsilon_0 > 0$, the algorithm A runs in $T^{1+\epsilon_0}$ time and uses $m = T^{\epsilon_0}$ random bits, and that the following hold:

$$x \in \Pi^{\mathsf{Y}} \implies \Pr_{r \in \{0,1\}^m}[A(x,r) = 1] \geq 1 - T^{-\omega(1)}$$

and

$$x \in \Pi^{\mathsf{N}} \implies \Pr_{r \in \{0,1\}^m}[A(x,r) = 1] \leq T^{-\omega(1)} \ .$$

In the following, we use $\vec{r} \in \{0,1\}^{m \times m}$ to denote a list of $\{r_i\}_{i \in [m]}$ such that $r_i \in \{0,1\}^m$. Now, consider a new promise problem Π_1, defined as follows:

$$\Pi_1^{\mathsf{Y}} := \left\{ (x, \vec{r}) : \Pr_{w \in \{0,1\}^m}[\exists i \text{ s.t. } A(x, r_i \oplus w) = 1] = 1 \right\}$$

[21] That is, for every T-time algorithm A and every polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n \in \mathbb{N}}$,the probability over $x \sim \mathbf{x}_n$ that $\Pr_r[A(x,r) = 1] \geq 1/2$ yet $A(x,s) = 0$ for all $s \in H(x)$ is negligible in n.

and

$$\Pi_1^{\mathsf{N}} := \left\{ (x, \vec{r}) : \Pr_{w \in \{0,1\}^m} [\exists i \text{ s.t. } A(x, r_i \oplus w) = 1] \leq 1/2 \right\} .$$

We denote the input length to Π_1 by $n' = |x| + m^2 = n + T^{2\epsilon_0}$. Note that we have $\Pi_1 \in prRTIME[T^{1+O(\epsilon_0)}(n')]$, via the following algorithm A': On input $(x, \vec{r})$, draw a random $w \sim \{0,1\}^m$, and output 1 if and only if there exists $i \in [m]$ such that $A(x, r_i \oplus w) = 1$. Hence, by our assumption, there is a deterministic algorithm B running in time $T_B = T^{1+O(\epsilon)}(n')$ that solves Π_1 with probability $1 - n^{-\omega(1)}$ over any polynomial-time samplable distribution.

Let H be a targeted hitting set that fools T_B-time algorithms, and let $M = T_B^{\epsilon}$ be the hypothesized bound on the number of output strings of H. Note that H is computable in time $T_B^{1+\epsilon}$, and in particular, in polynomial time. Our algorithm E now works as follows: Given an input $x \in \{0,1\}^n$, compute $\vec{r}^{(1)}, \ldots, \vec{r}^{(M)}$ from $H(x)$, output 1 if and only if there exists $i \in [M]$ such that $B(x, \vec{r}^{(i)}) = 1$.

We claim that E solves Π over any polynomial-time samplable distribution with probability $1 - n^{-\omega(1)}$. To see this, let $\mathcal{D}$ be a polynomial-time samplable distribution over $\{0,1\}^n$. Note that E fails to solve $x \sim \mathcal{D}$ if one of the following holds:

1. $x \in \Pi^{\mathsf{Y}}$ and for every $i \in [M]$ it holds that $B(x, \vec{r}^{(i)}) = 0$.
2. $x \in \Pi^{\mathsf{N}}$ and there exists $i \in [M]$ such that $B(x, \vec{r}^{(i)}) = 1$.

We use $\mathcal{E}_1$ and $\mathcal{E}_2$ to denote the two events above, and we bound their probability separately.

Bounding the probability of $\mathcal{E}_1$. Let $\mathcal{D}_1$ be the distribution obtained by drawing $x \sim \mathcal{D}$ and $\vec{r} \in \{0,1\}^{m \times m}$ and outputting $(x, \vec{r})$. By a standard argument, when $x \in \Pi^{\mathsf{Y}}$ it holds that $\Pr_{\vec{r}}[(x, \vec{r}) \in \Pi_1^{\mathsf{Y}}] \geq 1 - T^{-\omega(1)}$. Also note that B solves Π_1 over $\mathcal{D}_1$ with probability at least $1 - T^{-\omega(1)}$. It follows that

$$\Pr_{x \sim \mathcal{D}} \left[x \in \Pi^{\mathsf{Y}} \text{ and } \Pr_{\vec{r}}[B(x, \vec{r}) = 1] < 1/2 \right] < T^{-\omega(1)}.$$

From our assumption, it follows that:

$$\Pr_{x \sim \mathcal{D}} \left[\Pr_{\vec{r}}[B(x, \vec{r}) = 1] \geq 1/2 \text{ and } \bigwedge_{i \sim [M]} B(x, \vec{r}^{(i)}) = 0 \right] < T^{-\omega(1)} .$$

Combining the above two, we have that

$$\Pr_{x \sim \mathcal{D}} \left[x \in \Pi^{\mathsf{Y}} \text{ and } \bigwedge_{i \sim [M]} [B(x, \vec{r}^{(i)}) = 0 \right] < T^{-\omega(1)} .$$

Bounding the probability of $\mathcal{E}_2$. Now, we consider the second term. Consider a new distribution $\mathcal{D}_2$ obtained by drawing $x \sim \mathcal{D}$ and then outputting $(x, \vec{r}^{(i)})$ for a random $i \in [M]$.

Note that when $x \in \Pi^{\mathsf{N}}$, we have that $\Pr_{\vec{r}}[(x, \vec{r}) \in \Pi_1^{\mathsf{N}}] = 1$. Since B solves Π_1 over $\mathcal{D}_2$ with probability at least $1 - T^{-\omega(1)}$, it follows that

$$\Pr_{x \sim \mathcal{D},\ i \sim [M]} [x \in \Pi^{\mathsf{N}} \text{ and } B(x, \vec{r}^{(i)}) = 1] \leq T^{-\omega(1)} .$$

Taking a union bound over $i \in [M]$, we have

$$\Pr_{x \sim \mathcal{D}} [x \in \Pi^{\mathsf{N}} \text{ and } \exists i \in [M], \quad B(x, \vec{r}^{(i)}) = 1] \leq T^{-\omega(1)} .$$

When H uses advice. To handle advice, note that now B also needs $\alpha(T^{1+O(\epsilon_0)})$ bits of advice and T needs $\alpha(T_B)$ bits of advice. The analysis still works with straightforward adaptions. ■

3.1.2 OWFs suffice for derandomization over the uniform distribution

The following claim asserts that OWFs alone, without any additional assumption, suffice to obtain near-linear-time derandomization that succeeds on average over the *uniform distribution*. Note that this derandomization is fast (and, in particular, it bypasses the conditional impossibility results for worst-case derandomization in [CT21b; CT23b]), and that it errs rarely, i.e. with negligible probability $n^{-\omega(1)}$. However, we are not guaranteed that errors are infeasible to find.

The proof essentially follows from a lemma of Kinne, van Melkebeek, and Shaltiel [KMS12], and as far as we are aware it was not observed in writing before. For example, several previous results deduced the same conclusion from OWFs and *additional* assumptions (see [CT21b, Theorem 1.7] and [CT21a, Theorem 6.12]).

Claim 3.6. *Suppose that Assumption 3.7 is true. Then, for every polynomial T and constants $\epsilon > 0$ and $c > 1$ and every $\Pi_0 \in prBPTIME[T]$ there is $\Pi \in prDTIME[T^{1+\epsilon}]$ such that*

$$\Pr_{x\in\{0,1\}^n}\left[x \in \Pi_0^Y \setminus \Pi^Y \vee x \in \Pi_0^N \setminus \Pi^N\right] \leq n^{-\omega(1)} .$$

Proof. Let $\Pi_0 \in prBPTIME[T]$ and let A_0 be a probabilistic algorithm deciding Π_0 in time T. By naive error-reduction, we can assume that A_0 runs in time $T' = T^{1+o(1)}$ and has error at most $n^{-\omega(1)}$. By our hypothesis, there is a PRG G^{lin} with stretch $n^\epsilon \mapsto T'$ and running time at most $T^{1+\epsilon/2}$, and define $A(x) = A_0(x, G^{\mathsf{lin}}(x_{1,\ldots,n^{\epsilon/2}}))$.

Let L be any language that agrees with Π. Observe that $L \in \mathcal{P}/\mathrm{poly}$, and that by definition of A_0 we have $\Pr_{x\in\{0,1\}^n, r\in\{0,1\}^{T'}}[A_0(x,r) \neq L(x)] \leq n^{-\omega(1)}$. Using [KMS12, Lemma 1], we deduce that $\Pr_{x\in\{0,1\}^n}[A(x) \neq L(x)] \leq n^{-\omega(1)}$, relying on the fact that G^{lin} is an $n^{-\omega(1)}$-PRG for tests of the form $T(x,r) = A(x,r) \oplus L(x)$. The claim follows by defining Π to be the language that A decides, and observing that the event "$x \in \Pi_0^Y \setminus \Pi^Y \vee x \in \Pi_0^N \setminus \Pi^N$" is a subset of the event "$A(x) \neq L(x)$". ■

3.2 Near-linear-time PRG

In this section we present assumptions concerning PRGs that run in near-linear time, and have polynomial stretch. The first assumption was used in most prior works (see, e.g., [CT21b; CT21a; CT23b; DT23]), and it follows from the existence of OWFs:

Assumption 3.7 (near-linear-time PRG)**.** *The* near-linear-time PRG assumption *is that for every polynomial $T(n) \geq n$ and every $\epsilon > 0$, there is an algorithm G^{lin} that gets input $(1^n, s)$ where $s \in \{0,1\}^{T(n)^\epsilon}$, runs in time $T(n)^{1+\epsilon}$, and outputs $T(n)$ bits such that the following holds. For every large enough $n \in \mathbb{N}$ there is no $T^{-\omega(1)}$-distinguisher circuit of size $T(n)$ for $G^{\mathsf{lin}}(1^n, \cdot)$.*

Theorem 3.8 (OWFs $\Rightarrow$ near-linear-time PRGs; see, e.g., [CT21b, Theorem 4.3])**.** *If there are non-uniformly secure one-way functions, then Assumption 3.7 holds.*

We comment that the security of the PRG in Assumption 3.7 against circuits (and the security of the OWF in Theorem 3.8 against circuits) is not necessary for our results, and it could be replaced with security against uniform adversaries over all polynomial-time samplable distributions. We chose the formulation above merely for simplicity.

The second assumption that we will use is a generalization of the first one. Observe that the first assumption allows to stretch ℓ^ϵ bits to ℓ bits in time $T^{1+\epsilon}$ and in a way that fools T-time algorithms, for the specific value of $\ell = T$. The next assumption will generalize this to values of ℓ that may also be smaller than T; indeed, the PRG still runs in time $T^{1+\epsilon}$ and fools T-time algorithms, the only difference being that the stretch is $\ell^\epsilon \mapsto \ell$. Since small values of ℓ are allowed, we will need to be more careful, and avoid making a non-uniform assumption; we thus assume a PRG that works over all polynomial-time samplable distributions.

Assumption 3.9 (generalized near-linear-time PRG)**.** *The* generalized near-linear-time PRG assumption *is that for every polynomial $T(n) \geq n$ and every $\epsilon > 0$ and every $\ell(n) \geq (1+\epsilon) \cdot \log(T)$, there is an algorithm G_ℓ^{genlin} that gets input $(1^n, s)$ where $s \in \{0,1\}^{\ell(T(n))^\epsilon}$, runs in time $T(n)^{1+\epsilon}$, and outputs $\ell(T(n))$ bits such that the following holds. For every large enough $n \in \mathbb{N}$ and every uniform algorithm S running in time polynomial in T, the probability that $S(1^n)$ outputs a $1/10$-distinguisher circuit of size $T(n)$ for $G^{\mathsf{genlin}}(1^n, \cdot)$ is negligible in T.*

Assumption 3.9 will not be used in our main results, but as explained in Section 1.2.2, the assumption is useful to prove extensions of our results wherein we eliminate the small number of advice bits.

We stress that both Assumption 3.7 and Assumption 3.9 do not apparently yield derandomization that runs faster than sub-exponential time. The main crux of our proofs will be exponentially reducing the number of random coins that probabilistic machines use, without increasing their time overhead (and while only erring on inputs that are infeasible to find).

3.3 Standard constructions in pseudorandomness

We will need a version of the Nisan-Wigderson [NW94] generator from [CT21b] such that the generator is instantiated with very small output length, so that both the generator and reconstruction are very fast. In particular, for a truth-table of length N, the reconstruction receives oracle access to $\approx N^{.99}$ bits of advice, and runs in time $\approx N^{.01}$.

Theorem 3.10 (the NW generator with small output length; see [CT21b, Theorem 4.1])**.** *There is a universal constant $c_{\mathsf{NW}} > 1$ such that for all $\epsilon_{\mathsf{NW}} > 0$ there are two algorithms G_{NW} and R_{NW} that for any $N \in \mathbb{N}$ and any $f \in \{0,1\}^N$ satisfy the following:*

1. *(**Generator.**) The generator $G_{\mathsf{NW}}(f)$ runs in time $N^{1+4c_{\mathsf{NW}}^2\sqrt{\epsilon_{\mathsf{NW}}}}$ and outputs a list of $N^{\epsilon_{\mathsf{NW}}}$-bit strings.*
2. *(**Reconstruction.**) For any $(1/N^{\epsilon_{\mathsf{NW}}})$-distinguisher $D\colon \{0,1\}^{N^{\epsilon_{\mathsf{NW}}}} \to \{0,1\}$ for $G_{\mathsf{NW}}(f)$ there is a string adv of length $N^{1-c_{\mathsf{NW}}\cdot\sqrt{\epsilon_{\mathsf{NW}}}}$ such that the following holds. When R_{NW} gets input $i \in [N]$ and oracle access to D and to advice adv, it runs in time $N^{c_{\mathsf{NW}}\cdot\sqrt{\epsilon_{\mathsf{NW}}}}$ and outputs f_i.*

Moreover, there is a probabilistic oracle machine P_{NW} that for any $(1/N^{\epsilon_{\mathsf{NW}}})$-distinguisher $D\colon \{0,1\}^{N^{\epsilon_{\mathsf{NW}}}} \to \{0,1\}$ for $G_{\mathsf{NW}}(f)$, when given input 1^N and oracle access to f and to D satisfies the following. The machine P_{NW} runs in time $O(N^{1+c_{\mathsf{NW}}\cdot\epsilon_{\mathsf{NW}}})$, and with probability at least $2/3$ outputs adv such that $R_{\mathsf{NW}}^{D,\mathsf{adv}}(i) = f_i$ for all $i \in [N]$.

Proof sketch. The only part that did not explicitly appear in [CT21b, Theorem 4.1] is the algorithm P_{NW}. Since the details are standard following [NW94; STV01], we sketch the proof.

Recall that adv consists of an index $i \in [M]$ for a hybrid argument (where $M = N^{\epsilon_{\mathsf{NW}}}$), values for the seed of NW outside the i^{th} set in the design, a $(M-i)$-bit suffix of an input for D, a

list-decoding index $j \in [\mathrm{poly}(M)]$, and fixed randomness for the local list-decoder that G_{NW} is instantiated with (which is the code of [STV01]). In particular, when choosing adv uniformly at random, with probability at least $1/\mathrm{poly}(M)$ it satisfies the requirements and R^{D}_{NW} correctly computes f.

The machine P_{NW} repeatedly chooses adv, and for each choice it uses its oracle D to simulate $R^{D,\mathsf{adv}}_{\mathsf{NW}}$ on all inputs and compare to result to f (which P_{NW} also has as an oracle). Each attempt can be performed in time $N \cdot \mathrm{poly}(M)$, and after $\mathrm{poly}(M)$ attempts P_{NW} succeeds with high probability (and with zero error), with a total runtime of $N \cdot \mathrm{poly}(M)$. ■

We will also need a randomness-efficient sampler. For concreteness, we will use the extractor of Guruswami, Umans, and Vadhan [GUV09], and rely on the standard equivalence between randomness extractors and averaging samplers (see, e.g., [Vad12, Corollary 6.24]).

Definition 3.11 (averaging sampler)**.** *We say that* $\mathsf{Ext}\colon \{0,1\}^n \times [D] \to \{0,1\}^m$ *is an* (ϵ,δ)*-sampler if for every* $T \subseteq \{0,1\}^m$*, with probability at least* $1-\delta$ *over* $z \in \{0,1\}^n$ *it holds that* $\Pr_{i\in[D]}[\mathsf{Ext}(z,i) \in T] \in |T|/2^m \pm \epsilon$*.*

Theorem 3.12 (the extractor of [GUV09])**.** *For every constant* $\alpha > 0$ *and every time-computable* $k(n), \epsilon(n)$ *there is a polynomial-time algorithm that for every* $n \in \mathbb{N}$ *computes a* (k,ϵ)*-extractor*

$$\mathsf{Ext}\colon \{0,1\}^n \times [D] \to \{0,1\}^{(1-\alpha)\cdot k}\,,$$

where $D = \mathrm{poly}(n, 1/\epsilon)$*.*

Theorem 3.13 (seeded extractors are averaging samplers)**.** *Every* (k,ϵ)*-extractor* $\mathsf{Ext}\colon \{0,1\}^n \times [D] \to \{0,1\}^m$ *is an* (ϵ,δ)*-sampler, where* $\delta = 2^{k-n}$*.*

4 A superfast targeted generator

In this section we construct a new variant of the Chen-Tell generator with almost-linear running time. We first define the type of circuit families that we will work with.

Definition 4.1 (sufficiently uniform circuit families)**.** *Let* $T, d, \ell\colon \mathbb{N} \to \mathbb{N}$ *be computable in linear time such that* $T(n)$ *is always a power of* 2*. We say* $\{C_n\colon \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ *is a* sufficiently uniform *circuit family of width* $T(n)$ *and depth* $d(n)$*, if the following holds:*

- *The circuit has* $d+1$ *layers, each with exactly* T *gates. For convenience, we assume the first* n *gates on the first layer are the input gates, and the first* $\ell(n)$ *gates on the last layer are the output gates. We also assume that all the non-input gates on the first layer have value* 0*.*
- *All gates are of one of the following types:* NAND, AND, OR *gates of fan-in* 2*. For every non-input layer, the gates on that layer have the same type. All gates except for the last layer have fan-out* 2*. There is a* $\mathrm{polylog}(T)$*-time algorithm that takes* $i \in \{2,\dots,d+1\}$ *as input and outputs the type of gates on layer* i*.*
- *For every* $i \in \{2,\dots,d+1\}$*, the functions* $\mathsf{IN}_{i,0}\colon \{0,1\}^{\log T} \to \{0,1\}^{\log T}$ *and* $\mathsf{IN}_{i,1}\colon \{0,1\}^{\log T} \to \{0,1\}^{\log T}$ *that maps a gate index on layer* i *to the index of its first and second inputs can be computed by* $\mathcal{P}$*-uniform* $O(\mathrm{loglog}T)$ *depth circuits.*[22]

[22] In more detail, there is an algorithm that runs in $O(\mathrm{polylog}(T))$ time, takes $i \in \{2,\dots,d\}$ and $\alpha \in \{0,1\}$ as input, outputs an $O(\mathrm{loglog}T)$-depth circuit computing IN_α. The same holds for the $\mathsf{OUT}_{i,\alpha}$ below.

Similarly, for every $i \in [d]$ the functions $\mathsf{OUT}_{i,0}\colon \{0,1\}^{\log T} \to \{0,1\}^{\log T}$ and $\mathsf{OUT}_{i,1}\colon \{0,1\}^{\log T} \to \{0,1\}^{\log T}$ that maps a gate index on layer i to the index of its first and second output destination can be computed by $\mathcal{P}$-uniform $O(\log\log T)$ depth circuits.

The main theorem we will prove in this section is the following.

Theorem 4.2 (superfast Chen-Tell generator)**.** *Let $T, d, \ell\colon \mathbb{N} \to \mathbb{N}$ be computable in linear time such that $T(n)$ is always a power of 2. Let $\{C_n\colon \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ be a sufficiently uniform circuit family of width $T(n)$ and depth $d(n)$. Then, for every sufficiently small constant $\delta \in (0,1)$ there is a deterministic algorithm SPRG_C and a probabilistic oracle algorithm Rec_C that for every $z \in \{0,1\}^n$ they satisfy the following:*

1. **Somewhere Generator.** *When SPRG_C gets input $z \in \{0,1\}^n$ it runs in time $d \cdot T^{1+O(\sqrt{\delta})}$ and outputs $d' = d(n) \cdot \mathrm{polylog}(T)$ lists of M-bit strings, denoted by $L_z^{(1)}, \ldots, L_z^{(d')}$, where $M = T^\delta$.*
2. **Reconstruction.** *Suppose that Rec_C gets input $z \in \{0,1\}^n$ and oracle access to a function $D\colon \{0,1\}^M \to \{0,1\}$ such that the following holds:*
   - *For every $i \in [d']$,*

$$\left| \Pr_{r\in\{0,1\}^M}[D(r) = 1] - \Pr_{r\in\left[|L^{(i)}|\right]}[D(L_z^{(i)}[r]) = 1] \right| \geq 1/M\,,$$

   where $L_z^{(i)}[r]$ denotes the r-th element in the list $L_z^{(i)}$. Then $\mathsf{Rec}_C(z)$ runs in time $(d+n)\cdot T^{O(\sqrt{\delta})}$ and space $O(n) + T^{O(\sqrt{\delta})}$, makes at most $T^{O(\sqrt{\delta})}$ queries to D, and with probability at least $2/3$ outputs $C_n(z)$.

4.1 Polynomial decomposition

In this section we present a highly efficient polynomial decomposition of the computation of any sufficiently uniform circuit. This is essentially the same notion as the bootstrapping system that was described in Section 2.1. We begin with some notation.

Notation. Let $\mathbb{F}$ be a finite field. When we say a function $P\colon \mathbb{F}^m \to \mathbb{F}^k$ is a polynomial of maximum individual degree at most d, we simply mean that for every $i \in [k]$, the coordinate function $P_i(x) = P(x)_i$ is a polynomial of maximum individual degree at most d.

Lemma 4.3 (Polynomial decomposition)**.** *Let $T, d, \ell\colon \mathbb{N} \to \mathbb{N}$ be computable in linear time such that $T(n)$ is always a power of 2. Let $\{C_n\colon \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ be a sufficiently uniform circuit family of width $T(n)$ and depth $d(n)$. Let $h = h(n)$ and $m = m(n)$ be such that $h(n)$ is a power of 2, $\log T \leq h(n) \leq T$ and $h^m = 2T$. Let $\mathbb{F} = \mathbb{F}(n)$ be a finite field such that $|\mathbb{F}| \geq h$ and $|\mathbb{F}|$ is a power of 2.*

Then, there is a universal constant $c_1 > 1$ such that for every input $z \in \{0,1\}^n$, there is a list of polynomials $\{P_i^{(z)}\colon \mathbb{F}^m \to \mathbb{F}^{\log T+1}\}_{i\in[\tau]}$ where $\tau = d \cdot \mathrm{polylog}(T)$ such that the following holds (for brevity, below we use P_i to denote $P_i^{(z)}$ when there is no confusion):

- **(Arithmetic setting.)** *Each P_i has maximum individual degree at most $\Delta = h \cdot \mathrm{polylog}(T)$.*
- **(Input layer.)** *There is an algorithm that takes $z \in \{0,1\}^n$ and $x \in \mathbb{F}^m$ as input, and outputs $P_1(x)$ in $n \cdot h^{c_1}$ time and $O(n) + h^{c_1}$ space.*

- **(Output layer.)** *There is an algorithm that takes $i \in [\ell(n)]$ as input, makes a query to $P_\tau^{(z)}$, and outputs $C(z)_i$ in h^{c_1} time, for every $z \in \{0,1\}^n$.*
- **(Downward self-reducibility.)** *There is an algorithm that takes $i \in \{2,\dots,\tau\}$ and $x \in \mathbb{F}^m$ as input, makes at most h^{c_1} queries to P_{i-1}, and outputs $P_i(x)$ in h^{c_1} time.*
- **(Printing time of polynomials.)** *There is an algorithm that takes $i \in \{1,\dots,\tau\}$ and $z \in \{0,1\}^n$ as input, and prints the truth-table of $P_i^{(z)}$ in $O(\tau \cdot h^{c_1} \cdot |\mathbb{F}|^m)$ time.*

Proof. We begin with some notation.

Notation. From now on, we will assume $\mathbb{F}$ has characteristic number 2. Let $H \subseteq \mathbb{F}$ be an $\mathbb{F}_2$-subspace of $\mathbb{F}$ with size h. That is, there are $\xi_1,\dots,\xi_{\log h} \in \mathbb{F}$ such that $H = \left\{\sum_{i=1}^{\log h} z_i \cdot \xi_i : z \in \{0,1\}^{\log h}\right\}$. Note that in our constructions, we will explicitly pick an H such that the corresponding bases $\xi_1,\dots,\xi_{\log h} \in \mathbb{F}$ can be found efficiently (i.e., in $\widetilde{O}(\log h \cdot |\mathbb{F}|)$ time).

We then have a natural bijection $\kappa_h\colon \{0,1\}^{\log h} \to H$ where $\kappa_h(z) = \sum_{i=1}^{\log h} z_i \cdot \xi_i$ for $z \in \{0,1\}^{\log h}$. We can then construct another bijection $\kappa\colon \{0,1\}^{\log T+1} \to H^m$ by taking a direct product of κ_h (note that $2T = h^m$ implies $\log T + 1 = m \cdot \log h$).

Note that since $|\mathbb{F}|$ is a power of 2, by identifying $\{0,1\}^{\log T+1}$ with $\mathbb{F}_2^{\log T+1}$, κ is a group homomorphism. That is, for every $\alpha,\beta \in \mathbb{F}_2^{\log T+1}$, we have

$$\kappa(\alpha+\beta) = \kappa(\alpha) + \kappa(\beta)\ ,$$

where the additions on two sides are over $\mathbb{F}_2^{\log T+1}$ and over $\mathbb{F}^m$, respectively. This property will be crucial for us later in the constructions.

For every $\omega \in H$, let $e_\omega\colon \mathbb{F} \to \mathbb{F}$ be the degree-$(h-1)$ polynomial such that $e_\omega(\omega) = 1$ and $e_\omega(x) = 0$ for every $x \in H \setminus \{\omega\}$; e_ω can be constructed in $\mathrm{poly}(h)$ time via a standard interpolation.

In the proof, we will encode $[T]$ by $\{0,1\}^{\log T}$ via a natural bijection that maps $i \in [T]$ to the i-th lexicographically smallest string from $\{0,1\}^{\log T}$. When we say $g \in \{0,1\}^{\log T}$ is a gate index, we mean it denotes the (integer) index of a gate via this natural bijection.

Let $c_1 > 1$ be a sufficiently large universal constant. We say a list of polynomial $\{Q_1,\dots,Q_k\colon \mathbb{F}^m \to \mathbb{F}^*\}$ is downward self-reducible,[23] if there is an algorithm that takes $i \in \{2,\dots,k\}$ and $x \in \mathbb{F}^m$ as input, makes at most h^{c_1} queries to Q_{i-1}, and outputs $Q_i(x)$ in h^{c_1} time.

Let $L = \log T + 1$. We also need the following two lemmas, which will be used a lot in the rest of this subsection.

Lemma 4.4. *Let $C\colon \{0,1\}^M \to \{0,1\}$ be an $O(\log L)$-depth circuit, where M is a multiple of L. Given C as input, we can in $\mathrm{polylog}(T)$ time compute an arithmetic circuit $\widetilde{C}\colon \mathbb{F}^{M/L\cdot m} \to \mathbb{F}$ such that the following statements hold:*

- *For every $x \in \{0,1\}^M$, we have $C(x) = \widetilde{C}(\kappa^{\otimes(M/L)}(x))$.*[24]
- *$\widetilde{C}$ has maximum individual degree $\mathrm{polylog}(T) \cdot h$.*

[23] By F^* we mean that these polynomials can have different output lengths.

[24] Here, $\kappa^{\otimes(M/L)}(x)$ is obtained by first partitioning $x \in \{0,1\}^M$ into M/L blocks $x_1,\dots,x_{M/L} \in \{0,1\}^L$, then compute $\widetilde{x}_i = \kappa(x_i)$ for every $i \in [M/L]$, and finally output the concatenation $\widetilde{x}_1 \circ \widetilde{x}_2 \circ \cdots \circ \widetilde{x}_{M/L} \in \mathbb{F}^{(M/L)\cdot m}$.

Proof. We first arithmetize C in the standard way (replacing $\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}$ by their corresponding multi-linear extension polynomial) to obtain an arithmetic circuit $D\colon \mathbb{F}^M \to \mathbb{F}$ with individual degree at most $\mathrm{polylog}(T)$, in $\mathrm{polylog}(T)$ time.

Next, consider the inverse of the bijection $\kappa\colon \{0,1\}^m \to H$, $\tau = \kappa^{-1}$, that maps H to $\{0,1\}^m$.

We consider the extension of τ from H to $\mathbb{F}$, $\widetilde{\tau}\colon \mathbb{F} \to \mathbb{F}^m$, defined as follows:

$$\widetilde{\tau}(z) = \sum_{\omega \in H} \tau(\omega) \cdot e_\omega(z) \ .$$

Note that $\widetilde{\tau}$ and τ agrees on all inputs from H, and $\widetilde{\tau}$ has degree at most h.

Finally, for $z_1, \ldots, z_{M/L} \in \mathbb{F}^m$, we define

$$\widetilde{C}(z_1, \ldots, z_{M/L}) = D(\widetilde{\tau}(z_1), \ldots, \widetilde{\tau}(z_{M/L})) \ .$$

The two conditions follow directly from the construction of $\widetilde{C}$ and $\widetilde{\tau}$. ∎

Corollary 4.5. *Let $C\colon \{0,1\}^M \to \{0,1\}^L$ be an $O(\log L)$-depth circuit, where M is a multiple of L. Given C as input, we can in $\mathrm{polylog}(T)$ time compute an arithmetic circuit $\widetilde{C}\colon \mathbb{F}^{(M/L)\cdot m} \to \mathbb{F}^m$ such that the following statements hold:*

- *For every $x \in \{0,1\}^L$, we have $\kappa(C(x)) = \widetilde{C}(\kappa^{\otimes(M/L)}(x))$.*
- *$\widetilde{C}$ has maximum individual degree $\mathrm{polylog}(T) \cdot h$.*

Proof. For every $i \in [L]$, let $C_{[i]}\colon \{0,1\}^L \to \{0,1\}$ be the circuit that outputs the i-th output bit of C. We first apply lemma 4.4 to every $C_{[i]}$ to obtain an arithmetic circuit $D\colon \mathbb{F}^{(M/L)\cdot m} \to \mathbb{F}^m$. We then define

$$\widetilde{C}(z) = \kappa(D(z)) \ ,$$

where $z \in \mathbb{F}^{(M/L)\cdot m}$. ∎

In the following we will give several constructions of polynomial lists. Our final list of polynomials will be a concatenation of these constructions.

4.1.1 Construction: Degree Reduction List

First, we will make use of the standard linearization construction (introduced by [She92]) that reduces the maximum individual degree of a given polynomial from Δ to $h-1$ while preserving its values on H^m.

$\mathsf{DegReduction}(P)$

- **Input:** A polynomial $P\colon \mathbb{F}^m \to \mathbb{F}^\mu$ with maximum individual degree at most Δ and $\mu \leq \log T + 1$.
- **Output:** A list of polynomial $\mathsf{DegReductionList}(P) = \{Q_1, \ldots, Q_m\colon \mathbb{F}^m \to \mathbb{F}^\mu\}$ such that the following holds:
  1. For every $j \in [m]$, Q_j has maximum individual degree at most Δ.
  2. Q_m agrees with P on H^m and has maximum individual degree at most $h-1$.
  3. $\{P\} \circ \mathsf{DegReductionList}(P)$ is downward self-reducible.[a]

- We also set $\mathsf{DegReduction}(P) = Q_m$ to be the last element of $\mathsf{DegReductionList}(P)$.

[a] Here, $\{P\} \circ \mathsf{DegReductionList}(P)$ is the list obtained by adding P to the beginning of $\mathsf{DegReductionList}(P)$.

Let $Q_0 = P$. For every $i \in [m]$, we define $Q_i \colon \mathbb{F}^m \to \mathbb{F}^\mu$ by

$$Q_i(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_m) = \sum_{\omega \in H} e_\omega(x_i) \cdot Q_{i-1}(x_1, \ldots, x_{i-1}, \omega, x_{i+1}, \ldots, x_m).$$

Note that by the definition of e_ω, Q_i and Q_{i-1} agree on H^m. Moreover, by a simple induction, we can show for every $i \in [m]$, the maximum individual degree of variables $x_1, \ldots, x_i$ in Q_i is at most $h - 1$. Therefore, the maximum individual degree of Q_m is $h - 1$. The requirement that $P \circ \mathsf{DegReductionList}(P)$ is downward self-reducible follows from the definition of the Q_i.

4.1.2 Construction: Routing List

The next construction is the crux of our proof. We arithmetize the Bitonic sorter (a sorting network with $O(\log^2 n)$ depth) to "route" the gate values at layer i into correct places to feed the gates on layer $i + 1$.

$\mathsf{RoutingList}_i(P)$

- **Input:** A polynomial $P \colon \mathbb{F}^m \to \mathbb{F}$ with maximum individual degree at most $h - 1$ that represents the gate values at layer i. Formally, for every gate index $g \in \{0,1\}^{\log T}$, $P(\kappa(g \circ \alpha))$ equals the (Boolean) value of gate g for $\alpha \in \{0,1\}$).
- **Output:** A list of polynomials $\mathsf{RoutingList}_i(P) = \{Q_1, \ldots, Q_{\ell_{\mathsf{rt}}-1} \colon \mathbb{F}^m \to \mathbb{F}^{\log T+1}, Q_{\ell_{\mathsf{rt}}} \colon \mathbb{F}^m \to \mathbb{F}\}$ where $\ell_{\mathsf{rt}} = \mathrm{polylog}(T)$ such that the following holds:
  1. For every $j \in [\ell_{\mathsf{rt}}]$, Q_j has maximum individual degree at most Δ.
  2. $Q_{\ell_{\mathsf{rt}}}$ has maximum individual degree at most $h - 1$.
  3. For every $g \in \{0,1\}^{\log T}$ and $\alpha \in \{0,1\}$, $Q_{\ell_{\mathsf{rt}}}(\kappa(g \circ \alpha)) = P(\kappa(\mathsf{IN}_{i+1,\alpha}(g) \circ 0))$.
  4. $\{P\} \circ \mathsf{RoutingList}_i(P)$ is downward self-reducible.
- We also set $\mathsf{Routing}_i(P) = Q_{\ell_{\mathsf{rt}}}$ to be the last element of $\mathsf{RoutingList}_i(P)$.

Recall that $L = \log T + 1$. We will define $\mathsf{RoutingList}_i$ using the Bitonic sorter [Bat68], which is a construction of sorting network with $O(\log^2 n)$ depth. We first define several auxiliary polynomials to help us arithmetize this sorting network.

Comparison polynomials. Let $\mathsf{MAX} \colon \{0,1\}^L \times \{0,1\}^L \to \{0,1\}^L$ be the function that compares two input strings $\alpha, \beta \in \{0,1\}^L$ and outputs the larger one in lexicographic order. We also define $\mathsf{MIN} \colon \{0,1\}^L \times \{0,1\}^L \to \{0,1\}^L$ similarly. Note that $(\alpha, \beta) \mapsto (\mathsf{MIN}(\alpha, \beta), \mathsf{MAX}(\alpha, \beta))$ implements a comparator gate in a sorting network.

Both MAX and MIN can be implemented by $O(\log L)$-depth circuits. By Corollary 4.5, there are $\mathrm{polylog}(T)$-time computable polynomials $\widetilde{\mathsf{MIN}}, \widetilde{\mathsf{MAX}} \colon \mathbb{F}^L \to \mathbb{F}^L$ of $\mathrm{polylog}(T)$ maximum individual degree such that $\widetilde{\mathsf{MAX}}$ agrees with MAX on all Boolean inputs (similarly, $\widetilde{\mathsf{MIN}}$ agrees with MIN on all Boolean inputs).

Algorithm 1: Bitonic sorter

Input: An array arr of length n, indexed starting from 0
Output: The array arr sorted in place

```
for k = 2 to n with k ← 2k do
    for j = k/2 downto 1 with j ← j/2 do
        for i = 0 to n − 1 do
            ℓ = i ⊕ j
            if ℓ > i then
                if ((i & k) == 0 and arr[i] > arr[ℓ]) or ((i & k) ≠ 0 and arr[i] < arr[ℓ])
                 then
                    Swap arr[i] and arr[ℓ]
```

Figure 1: Pseudocode for the non-recursive implementation of the bitonic sorter.

Constructing the input polynomial to the sorting network. Given a polynomial $P\colon \mathbb{F}^m \to \mathbb{F}$ with maximum individual degree at most $h-1$ that represents the gate values at layer i.

Note that since for every $\alpha \in \{0,1\}$, $\mathsf{OUT}_{i,\alpha}$ can be computed by $O(\mathrm{loglog}T)$-depth circuits, there are polylog(T)-time computable polynomials $\widetilde{\mathsf{OUT}}_{i,\alpha}\colon \mathbb{F}^{\log T} \to \mathbb{F}^{\log T}$ of polylog(T) degree such that $\widetilde{\mathsf{OUT}}_{i,\alpha}$ agrees with $\mathsf{OUT}_{i,\alpha}$ on all Boolean inputs.

Via standard interpolation, we can construct polylog$(T)\cdot$poly(h)-time computable polynomials $\widetilde{\mathsf{POUT}}_i\colon \mathbb{F}^m \to \mathbb{F}^{\log T}$ with maximum individual degree $h\cdot$polylog(T) such that for every $g \in \{0,1\}^{\log T}$ and $\alpha \in \{0,1\}$, $\widetilde{\mathsf{POUT}}_i(\kappa(g\circ\alpha)) = \mathsf{OUT}_{i,\alpha}(g)$.

Now, we define the polynomial $\overline{W}_1(z) = \widetilde{\mathsf{POUT}}_i(z)\circ P(z)$ for $z\in\mathbb{F}^m$. Note that $\overline{W}_1\colon \mathbb{F}^m \to \mathbb{F}^L$ has maximum individual degree Δ. Also, for $g\in\{0,1\}^{\log T}$ and $\alpha\in\{0,1\}$, $\overline{W}_1(\kappa(g\circ\alpha)) = \mathsf{OUT}_{i,\alpha}(g)\circ P(\kappa(g\circ\alpha))$. That is, on H^m, $\overline{W}_1$ encodes a list of $2T$ pairs of a destination gate index and an outgoing value. Our goal is to sort them so that we can apply the gates of the next layer in a straightforward way.

Arithmetization of the sorting network. To arithmetize the Bitonic sorter, we recall its (non-recursive) pseudo-code below [Bat68]. In the description below, operation $\oplus$ and & denotes bit-wise XOR and bit-wise AND, respectively.

Now we apply the algorithm above to sort an array of length 2^L. Fix k and j in the algorithm above. In the loop for i, the algorithm partitions all indices into groups $\{i,\ell\}$ such that i and ℓ only differs at the $\log_2 j$-th bit. Assuming $i<\ell$, the algorithm makes sure $\mathsf{arr}[i] < \mathsf{arr}[\ell]$ if the $\log_2 k$-th bit of i is 0, and $\mathsf{arr}[i] > \mathsf{arr}[\ell]$ otherwise. We also note that the $\log_2 k$-th bit of i and ℓ are the same.

Let arr be the array before the start of this i-loop, and narr be the array after the end of this i-loop. Given the discussions above, we can define $\mathsf{narr}[i]$ as follows: for $i\in\{0,1\}^L$, we have

$$\mathsf{narr}[i] = \begin{cases} \mathsf{MIN}(\mathsf{arr}[i],\mathsf{arr}[i\oplus j]) & \Big[(i_{\log_2 k}=0) \textbf{ and } (i<(i\oplus j))\Big] \textbf{ or } \Big[(i_{\log_2 k}=1) \textbf{ and } (i>(i\oplus j))\Big] \\ \mathsf{MAX}(\mathsf{arr}[i],\mathsf{arr}[i\oplus j]) & \text{otherwise} \end{cases}$$

Let $\mathsf{Aux}_{k,j}\colon \{0,1\}^L \to \{0,1\}$ be such that

$$\mathsf{Aux}_{k,j}(i) = \Big[(i_{\log_2 k}=0) \textbf{ and } (i<(i\oplus j))\Big] \textbf{ or } \Big[(i_{\log_2 k}=1) \textbf{ and } (i>(i\oplus j))\Big] .$$

Since $\mathsf{Aux}_{k,j}$ can be computed in $O(\log L) = O(\mathrm{loglog} T)$ depth. By Lemma 4.4, we can get a $\mathrm{poly}(h) \cdot \mathrm{polylog}(T)$-time computable polynomial $\widetilde{\mathsf{Aux}}_{k,j} \colon \mathbb{F}^m \to \mathbb{F}$ with maximum individual degree $h \cdot \mathrm{polylog}(T)$ such that $\widetilde{\mathsf{Aux}}_{k,j}(\kappa(i)) = \mathsf{Aux}_{k,j}(i)$ for every $i \in \{0,1\}^L$.

Let $W_1 = \mathsf{DegReduction}(\overline{W}_1)$. Now we can implement the Bitonic sorter in $d_{\mathsf{rt}} = O(\log^2 T)$ steps as follows: for every $t \in [d_{\mathsf{rt}}]$, let $j = j_t$ and $k = k_t$ be the corresponding j and k for the t-th i-loop of the algorithm. Let $\gamma = \kappa(j)$ (here, we interpret j as a $\log L$-bit string via the natural bijection).

We set

$$\overline{W}_{t+1}(i) = \widetilde{\mathsf{Aux}}(i) \cdot \widetilde{\mathsf{MIN}}(W_t(i), W_t(i+\gamma)) + \left(1 - \widetilde{\mathsf{Aux}}(i)\right) \cdot \widetilde{\mathsf{MAX}}(W_t(i), W_t(i+\gamma)) \,.$$

In the above, we crucially make use of the fact that since κ is an $\mathbb{F}_2$-homomorphism, $W_t(i+\gamma)$ has the same maximum individual degree as W_t. Note that over H^m, $\overline{W}_{t+1}$ agrees with narr if arr encodes the values of W_t on H^m. Note that $\overline{W}_{t+1}$ has maximum individual degree at most $h \cdot \mathrm{polylog}(T) + (h-1) \cdot \mathrm{poly}(L) \le \Delta$. We then set $W_{t+1} = \mathsf{DegReduction}(\overline{W}_{t+1})$. By the property of this sorting network, we know that $W_{d_{\mathsf{rt}}}$ encodes the sorted length-$2T$ list of $2T$ pairs of a destination gate index and an outgoing value encoded in $\overline{W}_1$. Because each gate on layer $i+1$ has fan-in exactly 2, this means that for every $g \in \{0,1\}^{\log T}$, $W_{d_{\mathsf{rt}}}(\kappa(g \circ 0))$ and $W_{d_{\mathsf{rt}}}(\kappa(g \circ 0))$ are the values on two input wires to the gate g on layer $i+1$.

Now we are ready to define $\mathsf{RoutingList}_i(P)$ as

$$\mathsf{RoutingList}_i(P) = \bigcirc_{j \in [d_{\mathsf{rt}}+1]} \left(\overline{W}_j \circ \mathsf{DegReductionList}(\overline{W}_j)\right) .$$

We also modify the last element of $\mathsf{RoutingList}_i(P)$ by deleting its first $\log T$ outputs to get a polynomial from $\mathbb{F}^m \to \mathbb{F}$.

Verifying the conditions. Now we are ready to verify the properties of $\mathsf{RoutingList}_i(P)$. The third conditions follows from our discussions above on $W_{d_{\mathsf{rt}}}$ (which is the last element of $\mathsf{RoutingList}_i(P)$). The other three conditions follow directly from the properties of $\mathsf{DegReductionList}$ and the definition of the $\overline{W}_j$.

4.1.3 Construction: Evaluation List

Finally we construct a list of polynomials that evaluate gates on layer i.

$\mathsf{EvaluationList}_i(P)$

- **Input:** A polynomial $P \colon \mathbb{F}^m \to \mathbb{F}$ with maximum individual degree at most $h-1$ that represents the wire values coming to layer i (meaning, for every gate index $g \in \{0,1\}^{\log T}$, $P(\kappa(g \circ \alpha))$ for $\alpha \in \{0,1\}$ equal the (Boolean) wire values coming to the gate g).
- **Output:** A list of polynomials $\mathsf{EvaluationList}_i(P) = \{Q_1, \ldots, Q_{m+1} \colon \mathbb{F}^m \to \mathbb{F}\}$ such that the following holds:
  1. For every $j \in [m+1]$, Q_j has maximum individual degree at most Δ.
  2. For every $g \in \{0,1\}^{\log T}$ and $\alpha \in \{0,1\}$, $Q_{m+1}(\kappa(g \circ \alpha)) = \mathsf{GATE}(P(\kappa(g \circ 0)), P(\kappa(g \circ 1)))$, where $\mathsf{GATE} \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{NAND}\}$ is the gate type of layer i.

3. Q_{m+1} has maximum individual degree at most $h-1$.
4. $\{P\} \circ \mathsf{EvaluationList}_i(P)$ is downward self-reducible.

- We also set $\mathsf{Evaluation}(P) = Q_{m+1}$ to be the last element of $\mathsf{EvaluationList}_i(P)$.

Let $\gamma = \kappa(0^{\log T} \circ 1)$. It follows that for every $g \in \{0,1\}^{\log T}$ and $\alpha \in \{0,1\}$, $\kappa(g \circ (1-\alpha)) = \kappa(g \circ \alpha) + \gamma$. Let $\widetilde{\mathsf{AND}}, \widetilde{\mathsf{OR}}, \widetilde{\mathsf{NAND}} \colon \mathbb{F}^2 \to \mathbb{F}$ be the multi-linear extension of the functions $\mathsf{AND}, \mathsf{OR}, \mathsf{NAND} \colon \{0,1\}^2 \to \{0,1\}$. Given the layer index i, in $\mathrm{polylog}(T)$ time we can compute the gate type of this layer, we let $\widetilde{\mathsf{GATE}}$ be the corresponding multi-linear extension.

We then define $W \colon \mathbb{F}^m \to \mathbb{F}$ as

$$W(z) = \widetilde{\mathsf{GATE}}(P(z), P(z+\gamma)) \ .$$

Now, we set $\mathsf{EvaluationList}_i(P) = \{W\} \circ \mathsf{DegReductionList}(W)$. Note that W has maximum individual degree at most $(h-1) \cdot O(1) \leq \Delta$. All conditions can be verified straightforwardly.

4.1.4 The Final Construction

Let $\mathsf{Base} \colon \mathbb{F}^m \to \mathbb{F}$ be the interpolation of the input $z \in \{0,1\}^n$. That is, let $w_1, \ldots, w_n$ be the first n elements from $\{0,1\}^{\log T}$ and $\omega_{i,\alpha} = \kappa(w_i \circ \alpha)$ for every $i \in [n]$ and $\alpha \in \{0,1\}$.

For $\omega \in H^m$, we define the polynomial

$$\mathsf{EQ}_\omega(x) = \prod_{i=1}^{m} e_{\omega_i}(x_i)$$

where $x \in \mathbb{F}^m$, note that EQ_ω has maximum individual degree $h-1$ and it is computable in $\mathrm{poly}(h)$ time. Note that for all $z \in H^m$, $\mathsf{EQ}_\omega(z) = 1$ if $z = \omega$ and 0 otherwise.

Then, Base is defined as

$$\mathsf{Base}(x) = \sum_{i=1}^{n} \sum_{\alpha \in \{0,1\}} z_i \cdot \mathsf{EQ}_{\omega_{i,\alpha}}(x) \ .$$

We set $V_1 = \mathsf{Base}$. For every $i \in \{2, \ldots, d+1\}$, we define $V_i = \mathsf{Evaluation}_i(\mathsf{Routing}_{i-1}(V_{i-1}))$.

The list is then

$$V_1 \circ \bigcirc_{i \in [d]} \left(\mathsf{RoutingList}_i(V_i) \circ \mathsf{EvaluationList}_{i+1}(\mathsf{Routing}_i(V_i))\right) \ .$$

We note that by the definitions of the V_i, $\mathsf{RoutingList}_i$, and $\mathsf{EvaluationList}_i$, the list above is downward self-reducible and all polynomials in it have maximum individual degree Δ. The conditions on input/output layers follow directly from the definition of V_1 and V_{d+1}. We also remark that some polynomials in the list may have less than $\log T + 1$ outputs, but we can add dummy outputs that always outputs 0 to make sure all polynomials always have $\log T + 1$ outputs.

Finally, we verify the printing time of the polynomials. Computing the truth-table directly from the definition of Base takes at least $n \cdot |\mathbb{F}|^m$ time, so we need a faster algorithm.[25] Let $B_0 \colon H^m \to \{0,1\}$ be the function such that, for every $i \in [n]$ and $\alpha \in \{0,1\}$, we have $B_0(\kappa(w_i, \alpha)) =$

[25] Indeed we just need an almost-linear time encoding algorithm for Reed-Muller codes, which is standard. We include a proof here just for completeness.

z_i, and all other entries of B_0 are set to 0. From its definition, we know that Base agrees with B_0 on H^m. Since Base has maximum individual degree $|H|-1$, Base is indeed the unique polynomial extension of B_0 from H^m to $\mathbb{F}^m$ maximum individual degree $|H|-1$.

Therefore we can compute the truth-table of Base as follows: For each $i \in [m]$, we define a function $B_i \colon \mathbb{F}^i \times H^{m-i} \to \mathbb{F}$ such that

$$B_i(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_m) = \sum_{\omega \in H} e_\omega(x_i) \cdot B_{i-1}(x_1,\ldots,x_{i-1},\omega,x_{i+1},\ldots,x_m) \ ,$$

where $x_1,\ldots,x_i \in \mathbb{F}$ and $x_{i+1},\ldots,x_m \in H$. We can see that B_m agrees with B_0 on H^m and B_m is a polynomial with maximum individual degree at most $h-1$. Therefore, B_m is the truth-table of Base. The algorithm above runs in $h^{c_1} \cdot |\mathbb{F}|^m$ time and $h^{c_1} + O(|\mathbb{F}|^m)$ space.

Finally, the truth-table of polynomials $P_i^{(z)}$ for $i > 1$ can be computed via the downward self-reducibility (first compute the truth-table of $P_2^{(z)}$, then $P_3^{(z)}$, and so on). The total running time can thus be bounded by $\tau \cdot h^{c_1} \cdot |\mathbb{F}|^m$. ■

4.2 From polynomial decomposition to a targeted generator

In this section we use the polynomial decomposition to prove Theorem 4.2. We first present some necessary standard technical tools, in Section 4.2.1, and then present the proof itself, in Section 4.2.2.

4.2.1 Standard technical tools

The following statements present versions of standard technical tools: The Nisan-Wigderson generator [NW94], the Goldreich-Levin local list-decoder of the Hadamard code [GL89], the local list-decoder for the Reed Muller code of Sudan, Trevisan, and Vadhan [STV01], and the local unique decoder for the Reed-Muller code.

The proof in Section 4.2.2 will use these tools. Readers who are familiar with these tools can safely skip the precise technical statements, and jump to Section 4.2.2.

Theorem 4.6 (the PRG of [NW94] with small output length and reconstruction as a learning algorithm; see, e.g., [CT21a, Theorem A.4])**.** *There exists a universal constant c_{NW} such that for every sufficiently small constant $\epsilon_{\mathsf{NW}} > 0$ there exist an oracle machine G and a probabilistic oracle machine R that satisfy the following:*

- **Generator:** *When given input $f \in \{0,1\}^N$, the machine G runs in time $N^{1+c_{\mathsf{NW}}\cdot\epsilon_{\mathsf{NW}}}$ and outputs $2^{\ell_{\mathsf{NW}}(N)}$ strings in $\{0,1\}^{N^{\epsilon_{\mathsf{NW}}}}$, where $\ell_{\mathsf{NW}}(N) = (1 + c_{\mathsf{NW}} \cdot \epsilon_{\mathsf{NW}}) \cdot \log(N)$.*
- **Reconstruction:** *When given input N (i.e., in binary) and oracle access to f, the machine R runs in time $N^{c_{\mathsf{NW}}\cdot\epsilon_{\mathsf{NW}}}$ and prints an oracle circuit C such that the following holds. For every $N^{-\epsilon_{\mathsf{NW}}}$-distinguisher $D \colon \{0,1\}^{N^{\epsilon_{\mathsf{NW}}}} \to \{0,1\}$ for $G(f)$, with probability at least $N^{-\epsilon_{\mathsf{NW}}}$ over the randomness of R it holds that $\Pr_{x\in[N]}[C^D(x) = f(x)] \geq 1/2 + N^{-3\epsilon_{\mathsf{NW}}}$.*

Theorem 4.7 (local list-decoding of the Hadamard code [GL89]; see, e.g., [CT21a, Theorem 4.9])**.** *There is a universal constant $c_2 > 1$ such that the following holds. Let $\epsilon \colon \mathbb{N} \to (0,1)$ and $a \colon \mathbb{N} \to \mathbb{N}$ be time-computable functions. Then, there exists a probabilistic algorithm (local list-decoder)* Dec *that gets input 1^{ℓ_0}, runs in time $\mathrm{poly}(\ell_0/\epsilon)$ and outputs an oracle circuit C that satisfies the following. For every function $P \colon \{0,1\}^{\ell_0} \to \{0,1\}^{a(\ell_0)}$, and every oracle $\widetilde{\mathsf{Had}}(P) \colon \{0,1\}^{\ell_0+a(\ell_0)} \to \{0,1\}$ that agrees with $\mathsf{Had}(P)$ on at least $1/2+\epsilon(\ell_0)$ of the inputs, with probability at least ϵ^{c_2} over the random coins of* Dec *it holds that $\Pr_{x\in\{0,1\}^{\ell_0}}\left[C^{\widetilde{\mathsf{Had}}(P)}(x) = P(x)\right] \geq \epsilon^{c_2}$.*

Theorem 4.8 (local list-decoding of the Reed-Muller code [STV01]; see, e.g., [Vad12, Section 7.6.3]). *There is a constant $c_3 > 1$ and a probabilistic algorithm (local list-decoder) that acts as follows. The algorithm gets as input a representation of a field $\mathbb{F}$ and parameters m, Δ, and gets oracle access to $\tilde{P}\colon \mathbb{F}^m \to \mathbb{F}$ that agrees with a degree-Δ polynomial $P\colon \mathbb{F}^m \to \mathbb{F}$ on at least $c_3 \cdot \sqrt{\Delta/|\mathbb{F}|}$ of the inputs. The algorithm runs in time $(|\mathbb{F}|, m)^{c_3}$, and outputs a circuit C such that with probability at least $1/(c_3 \cdot |\mathbb{F}|)$ it holds that $C^{\tilde{P}}(x) = P(x)$ for all $x \in \mathbb{F}^m$.*

Theorem 4.9 (local unique decoding of the Reed-Muller code; see, e.g., [AB09, Theorem 19.19]). *There is a constant $c_4 > 1$ and a probabilistic algorithm (local decoder) that acts as follows. The algorithm gets as input a representation of a field $\mathbb{F}$ and parameters m, Δ, η, runs in time $(|\mathbb{F}| \cdot m \cdot \log(1/\eta))^{c_4}$, and outputs an oracle circuit C. For every $\tilde{P}\colon \mathbb{F}^m \to \mathbb{F}$ that agrees with a degree-Δ polynomial $P\colon \mathbb{F}^m \to \mathbb{F}$ on at least $1 - (1 - \Delta/|\mathbb{F}|)/6$ of the inputs, with probability at least $1 - \eta$ it holds that $C^{\tilde{P}}(x) = P(x)$ for all $x \in \mathbb{F}^m$.*

4.2.2 Proof of Theorem 4.2

Let $c > 1$ be a sufficiently large universal constant (the constraints on c will be pointed out throughout the proof), and let $\epsilon = c \cdot \sqrt{\delta}$ be sufficiently small (relying on the assumption that $\delta > 0$ is sufficiently small). We first describe the generator, and then the reconstruction algorithm.

Generator. Given input $z \in \{0,1\}^n$, let $P_1, ..., P_{d'}$ be the polynomial decomposition of $C_n(z)$ from Lemma 4.3, with $h = T^{\epsilon}$ and $|\mathbb{F}| = h^{1+\epsilon}$, where $d' = d \cdot \mathrm{polylog}(T)$. For this parameter setting we have that $m = (1/\epsilon) \cdot (1 + 1/\log(T))$ and $|\mathbb{F}|^m = 2T \cdot T^{\epsilon^2 \cdot m} < T^{1+2\epsilon}$.

For each $i \in \{2, ..., d'\}$, the generator acts as follows. Let $f_i\colon \{0,1\}^{\ell'} \to \{0,1\}$ be the Hadamard encoding of P_i, where $\ell' = (m+1) \cdot \log(|\mathbb{F}|)$; that is, f_i parses its input $x \in \{0,1\}^{(m+1)\cdot\log(|\mathbb{F}|)}$ as $x_0 \in \mathbb{F}^m$ and $r \in \mathbb{F}$ and outputs $\langle P_i(x_0), r\rangle = \sum_{j \in [\log(|\mathbb{F}|)]} P_i(x_0)_j \cdot r_j \bmod 2$, where $P_i(x_0)_j$ (resp., r_j) is the j^{th} bit in the binary representation of $P_i(x_0)$ (resp., of r). The generator computes the truth-table of f_i, and gives it as input to G from Theorem 4.6 (instantiated with parameter $\epsilon_{\mathsf{NW}} = \delta$). The list L_i is the output list of $G(f_i)$.

Let us bound the running time of the generator. For $i \in \{2, ..., d'\}$, computing the truth-table of P_i can be done in time $d' \cdot |\mathbb{F}|^m \cdot h^{c_1} < d \cdot T^{1+2c_1\cdot\epsilon}$ (by Lemma 4.3), and transforming this truth-table to the truth-table of f_i can be done in time $O(|\mathbb{F}|^m \cdot |\mathbb{F}|) < T^{1+3c_1\cdot\epsilon}$. The generator G from Theorem 4.6 runs in time $2^{(1+c_{\mathsf{NW}}\cdot\epsilon_{\mathsf{NW}})\cdot\ell'} < (T^{1+3c_1\cdot\epsilon})^{1+c_{\mathsf{NW}}\cdot\epsilon_{\mathsf{NW}}} < T^{1+4c_1c_{\mathsf{NW}}\cdot\epsilon}$. Accounting for the $d' = d \cdot \mathrm{polylog}(T)$ iterations, the total running time of the generator is at most $d \cdot T^{1+c\cdot\epsilon}$.

Reconstruction. Given input $z \in \{0,1\}^n$ and oracle access to D as in the hypothesis. The reconstruction algorithm works in phases: For each phase $i \in \{2, ..., d'\}$, the algorithm will compute a circuit C_i of size $T^{(c/8)\cdot\epsilon}$ such that C_i^D computes the polynomial P_i.

The base case $i = 2$ and the inductive step $i > 2$ are essentially identical (with only one difference poined out below), so we now describe a generic phase for a fixed $i \in \{2, ..., d'\}$. Throughout the step, we will repeatedly use the following fact:

Fact 4.9.1. *Given any input $x = (x_0, r) \in \{0,1\}^{\ell'}$ and access to the oracle D, we can compute $f_i(x)$ in time $\begin{cases} n \cdot T^{(c/4)\cdot\epsilon} & i = 2 \\ T^{(c/4)\cdot\epsilon} & o.w. \end{cases}$ and in space at most $O(n) + T^{(c/4)\cdot\epsilon}$.*

Proof. We first compute $P_i(x_0)$ using the downward self-reducibility of the polynomial decomposition, which runs in time $h^{c_1} = T^{c_1\cdot\epsilon}$. Whenever the downward self-reducibility queries P_{i-1},

we answer as follows: When $i > 2$, we use the circuit C_{i-1} and the oracle D, which takes time $T^{c_1 \cdot \epsilon} \cdot \tilde{O}(T^{(c/8)\cdot\epsilon}) < T^{(c/4)\cdot\epsilon}$; and when $i = 2$ we use the algorithm for the input polynomial P_1, which uses time $n \cdot h^{c_1} < n \cdot T^{(c/4)\cdot\epsilon}$ and space $O(n) + h^{c_1} < n + T^{(c/4)\cdot\epsilon}$. We output $\langle P_i(x_0), r\rangle$. □

Description of each iterative step. The i^{th} phase in the reconstruction is as follows:

1. We run the **NW reconstruction** R from Theorem 4.6 for $t_{\mathsf{NW}} = O(\log(d') \cdot 2^{\epsilon_{\mathsf{NW}} \cdot \ell'})$ times. Whenever R queries f_i at point $x = (x_0, r)$ we use Fact 4.9.1 to answer. The t_{NW} iterations yield a list of oracle circuits, denoted by $C_i^{(1)}, ..., C_i^{(t_{\mathsf{NW}})}$.
2. For each $j \in [t_{\mathsf{NW}}]$, we run the **local list-decoder for the Hadamard code** with parameter $\epsilon_{\mathsf{GL}} = 2^{-3\epsilon_{\mathsf{NW}} \cdot \ell'}$, for $t_{\mathsf{GL}} = O(\log(d')/\epsilon_{\mathsf{GL}}^{c_2})$ times.[26] For each $j \in [t_{\mathsf{NW}}]$ and $j' \in [t_{\mathsf{GL}}]$, the decoder outputs a circuit C_{GL}, and we denote by $C_i^{(j,j')}$ the circuit that executes C_{GL} while answering its oracle queries using $C_i^{(j)}$ (i.e., $C_i^{(j,j')} = C_{\mathsf{GL}}^{C_i^{(j)}}$). This yields a list of oracle circuits $\left\{C_i^{(j,j')}\right\}_{j\in[t_{\mathsf{NW}}],j'\in[t_{\mathsf{GL}}]}$.
3. For each $(j, j') \in [t_{\mathsf{NW}}] \times [t_{\mathsf{GL}}]$, we run the **local list-decoder of the Reed-Muller code** with parameter Δ, for $t_{\mathsf{RM}} = O(\log(d') \cdot |\mathbb{F}|)$ times (see Theorem 4.8). For each (j, j', j'') (where $j'' \in [t_{\mathsf{RM}}]$), the decoder outputs a circuit C_{RM}, and we denote by $C_i^{(j,j',j'')}$ the circuit that executes C_{RM} while answering its queries using $C_i^{(j,j')}$. This yields a list $\left\{C_i^{(j,j',j'')}\right\}_{(j,j',j'')}$.
4. Now we **weed the list** of candidate circuits to find a circuit that agrees with P_i on at least 0.99 of the inputs. For $t = t_{\mathsf{NW}} \cdot t_{\mathsf{GL}} \cdot t_{\mathsf{RM}}$, we identify $[t]$ with the set of triplets (j, j', j''). For each $k = (j, j', j'') \in [t]$, we uniformly sample $\ell_t = O(\log(t) \cdot \log(d'))$ points $x_1, ..., x_{\ell_t} \in \mathbb{F}^m$ and compute $v_k = \Pr_{u\in[\ell_t]}\left[C_k^D(x_u) = P_i(u)\right]$ (where we compute P_i using Fact 4.9.1, and use our oracle access to D to compute C_k^D). Let $k^* = \text{argmax}\left\{v_k\right\}_{k\in[t]}$.
5. We run the **local (unique) decoder for the Reed-Muller code**, with confidence parameter $\eta = 1/(c \cdot d')$ (see Theorem 4.9). It outputs an oracle circuit C_{uRM}, and our final circuit C_i is an oracle circuit C_i that implements C_{uRM} while resolving its queries using $C_i^{(k^*)}$.

After all phases are complete, the algorithm has a circuit $C_{d'}$ such that (hopefully) $C_{d'}^D$ computes $P_{d'}$ exactly. We execute the output layer algorithm with $C_{d'}$ (and oracle D) on inputs $1, ..., \ell(n)$, we output the corresponding $\ell(n)$-bit string.

Complexity analysis. Let us first bound the running time of computing C_i (i.e., of a single phase). The NW decoder runs in time $2^{c_{\mathsf{NW}} \cdot \epsilon_{\mathsf{NW}} \cdot \ell'}$, the local list-decoders of Had and of RM run in times $(\log(|\mathbb{F}|)/\epsilon_{\mathsf{GL}})^{c_2}$ and $|\mathbb{F}|^{c_3}$ (respectively), and thus the size of each $C_i^{(j,j',j'')}$ is at most $2^{c_{\mathsf{NW}} \cdot \epsilon_{\mathsf{NW}} \cdot \ell'} \cdot (|\mathbb{F}|/\epsilon_{\mathsf{GL}})^{c_2+c_3}$. Denoting the time it takes to compute f_i (using Fact 4.9.1) by T_i, the running time of the i^{th} phase is at most

$$\begin{aligned} t_{\mathsf{NW}} \cdot t_{\mathsf{GL}} \cdot t_{\mathsf{RM}} \cdot t_\ell \cdot O\left(2^{c_{\mathsf{NW}} \cdot \epsilon_{\mathsf{NW}} \cdot \ell'} \cdot (|\mathbb{F}|/\epsilon_{\mathsf{GL}})^{c_2+c_3} + T_i\right) &< T^{4\epsilon+6c_2\epsilon+2\epsilon} \cdot \left(T^{6c_{\mathsf{NW}}c_2c_3\cdot\epsilon} + T_i\right) \\ &< T^{c/2\cdot\epsilon} \cdot T_i \,, \end{aligned}$$

[26] Specifically, we use Theorem 4.7 while considering P_i as a Boolean function and f_i as its Hadamard encoding. The domain of P_i as a Boolean function is of length $\ell_0 = m \cdot \log(|\mathbb{F}|)$ and its range is of length $a(\ell_0) = \log(|\mathbb{F}|)$. The constant $c_2 > 1$ is the universal constant from Theorem 4.7.

where we used the fact that $2^{\ell'} < T^{1+3c_1\epsilon} < T^2$ for a sufficiently small ϵ.

Since there are $d' - 1 < d \cdot \mathrm{polylog}(T) < d \cdot T^{\epsilon}$ phases, and using the bound on T_i (i.e., $T_i = n \cdot T^{c_1 \cdot \epsilon}$ when $i = 2$, and $T_i = T^{c_1 \cdot \epsilon}$ otherwise), the total running time of the reconstruction algorithm is at most $(d + n) \cdot T^{c \cdot \epsilon}$.

For space complexity, observe that each iteration i can be modeled as a time-$T^{(c/2)\cdot\epsilon}$ algorithm with oracle access to D and to C_{i-1}. Resolving each query to C_{i-1} takes space $O(n) + T^{(c/4)\cdot\epsilon}$ (using Fact 4.9.1), but this space can be discarded and reused after resolving each query. Thus, the algorithm runs in space at most $O(T^{(c/2)\cdot\epsilon} + n)$.

Correctness. Finally, let us argue correctness. We claim that for each of the $d' - 1$ phases, with probability at least $1 - 1/O(d')$ it holds that C_i^D computes P_i. This follows by a simple union-bound on the error probabilities of the individual steps in the phase, as follows:

1. By Theorem 4.6, with probability at least $1 - 1/O(d')$ there is $j \in [t_{\mathsf{NW}}]$ such that $(C_i^{(j)})^D$ agrees with f_i on at least $1/2 + 2^{-3\epsilon_{\mathsf{NW}} \cdot \ell'}$ of the inputs.
2. By Theorem 4.7 with the choice of $\epsilon_{\mathsf{GL}} = 2^{-3\epsilon_{\mathsf{NW}} \cdot \ell'}$, with probability at least $1 - 1/O(d')$ there is $j' \in [t_{\mathsf{GL}}]$ such that $(C_i^{(j,j')})^D$ agrees with P_i on at least $2^{-3c_2\epsilon_{\mathsf{NW}} \cdot \ell'}$ of the inputs.
3. To argue correctness using Theorem 4.8, we need to verify that $2^{-3c_2 \cdot \epsilon_{\mathsf{NW}} \cdot \ell'} > c_2 \cdot \sqrt{\Delta/|\mathbb{F}|}$, which holds due to our choice of $\epsilon = \Theta(\sqrt{\delta})$.[27] Hence, with probability at least $1 - 1/O(d')$ there is $j'' \in [t_{\mathsf{RM}}]$ such that $C_i^{(j,j',j'')}$ computes P_i exactly.
4. With probability at least $1 - o(1)$ it holds that $C_i^{(k^*)}$ agrees with P_i on at least $0.99 > 1 - (1 - \Delta/|\mathbb{F}|)/6$ of the inputs, and in this case with probability at least $1 - 1/O(d')$ it holds that C_i^D computes P_i exactly.

Union-bounding over all $d' - 1$ phases, with probability at least $2/3$ the final circuit $C_{d'}$ computes $P_{d'}$ exactly when given oracle access to D, in which case the output of the reconstruction algorithm is correct.

5 Superfast derandomization from hardness over all polytime samplable distributions

In this section we prove Theorem 1.2 Theorem 1.3 and Theorem 1.4. The proof of the first two appear in Sections 5.1 and 5.2.1 and rely on the generator construction from Section 4. The proof of Theorem 1.4 relies on the Nisan-Wigderson generator, and appears in Section 5.2.2.

5.1 Hardness for uniform medium-depth circuits

The following result is the main one needed to prove Theorem 1.2. The result assumes hardness for a fixed time bound T, and deduces derandomization of $pr\mathcal{RTIME}[T]$ (i.e., derandomization of algorithms with one-sided error). After proving the result we explain how Theorem 1.2 follows, which is basically a black-box application of Theorem 3.5.

[27] Specifically, recall that $\Delta = h \cdot \mathrm{polylog}(T)$, and thus $c_2 \cdot \sqrt{\Delta/|\mathbb{F}|} < h^{-\epsilon/3} = T^{-\epsilon^2/6}$. On the other hand, $2^{\ell'} < T^2$ and $\epsilon_{\mathsf{NW}} = \delta$, so $2^{-3c_2\epsilon_{\mathsf{NW}} \cdot \ell'} > T^{-6c_2\delta}$. Thus, for $\epsilon = c \cdot \sqrt{\delta}$ the condition is satisfied.

Theorem 5.1 (superfast and free-lunch derandomization from hardness on all polytime samplable distributions)**.** *For every $\epsilon > 0$ there is $\delta = \Theta(\epsilon)$ such that the following holds. Suppose that the near-linear-time PRG assumption is true. Let $T(n) \geq n$ be a polynomial, let $\bar{T} = T^{1+\delta}$, and let $f\colon \{0,1\}^* \to \{0,1\}^*$ be computable by a sufficiently uniform family of circuits of size $\bar{T}(n)$ and depth $d(n) = n^{o(1)}$. Assume that for every probabilistic algorithm A running in time $\bar{T}^{1-\delta/2}$, and every polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n \in \mathbb{N}$, we have that $\Pr_{x\sim\mathbf{x}_n}[A(x) = f(x)] \leq n^{-\omega(1)}$. Then,*

$$prRTIME[T] \subseteq \text{heur-}prDTIME[T^{2+\epsilon}]$$
$$prRTIME[T] \subseteq \text{heur-}prDTIME[T^{1+\epsilon}]/\tilde{O}(\log n)\,.$$

Furthermore, if we assume the generalized near-linear-time PRG (which is stronger than the near-linear-time PRG), then

$$prRTIME[T] \subseteq \text{heur-}prDTIME[T^{1+\epsilon}]\,.$$

Proof. Let $\Pi = (\Pi^Y, \Pi^N) \in prRTIME[T]$, let A be a probabilistic time-T algorithm deciding Π, and let $\epsilon > 0$. Let $\delta = \Theta(\epsilon)$ be sufficiently small, and let $\delta_{\mathsf{NW}} = \Theta(\delta^2)$ be sufficiently small. Define A' that gets input $x \in \{0,1\}^n$ and random coins $s \in \{0,1\}^{T^{\delta_{\mathsf{NW}}}}$, and outputs $A(x, G^{\mathsf{lin}}(s))$. Note that A' decides Π, since for any $x \in \{0,1\}^n$ it holds that $\Pr_r[A(x,r) = 1] \approx_{1/10} \Pr_s[A'(x,s) = 1]$ (this is true since G^{lin} $(1/10)$-fools the distinguisher $D_x(u) = A(x,u)$, which implies that if $x \in \Pi^Y$ there is s such that $D_x(G^{\mathsf{lin}}(s)) = 1$). Also note that A' runs in time $O(T^{1+\delta_{\mathsf{NW}}})$.

Derandomization in quadratic time. Given input $x \in \{0,1\}^n$, we use the targeted generator from Theorem 4.2 with the function f and with output length $M = T^{\delta_{\mathsf{NW}}}$. The generator outputs $\bar{T}^{1+o(1)}$ lists of $T^{\delta_{\mathsf{NW}}}$-bit strings, and their union yields a set $W(x)$ of $d \cdot \bar{T}^{1+O(\sqrt{\delta_{\mathsf{NW}}})}$ strings. We output $A''(x) = \vee_{w\in W(x)} A'(x,w)$. Note that A'' runs in time at most $\bar{T}^{1+O(\sqrt{\delta_{\mathsf{NW}}})} \cdot T^{1+\delta_{\mathsf{NW}}} < T^{(1+\delta)(1+O(\sqrt{\delta_{\mathsf{NW}}}))+1+\delta_{\mathsf{NW}}} < T^{2+\epsilon}$.

Assume towards a contradiction that there is a polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ such that for infinitely many $n \in \mathbb{N}$, with noticeable probability over $x \sim \mathbf{x}_n$ it holds that $x \in \Pi^Y$ and $A''(x) = 0$. For every such fixed x, the function $D_x(u) = A'(x,u)$ satisfies $\Pr_{r\in\{0,1\}^{T^{\delta_{\mathsf{NW}}}}}[D_x(r)] \geq 1/2$ but D_x rejects all strings in $W(x)$. In particular, D_x is a $(1/M)$-distinguisher for each list that the generator outputs. Thus, when we use the algorithm Rec from Theorem 4.2 and answer its queries using $D_x(u) = A'(x,u)$, we obtain an algorithm $R = \mathsf{Rec}^{D_x}$ that prints $f(x)$ (with probability at least $2/3$) in time

$$\begin{aligned}(d+n)\cdot \bar{T}^{O(\sqrt{\delta_{\mathsf{NW}}})} + \bar{T}^{O(\sqrt{\delta_{\mathsf{NW}}})}\cdot T^{1+\delta_{\mathsf{NW}}} &< n\cdot \bar{T}^{\delta/4} + \bar{T}^{\delta/4}\cdot T^{1+\delta_{\mathsf{NW}}} \\ &= O(\bar{T}^{(1+\delta_{\mathsf{NW}})/(1+\delta)+\delta/4}) \\ &< \bar{T}^{1-\delta/2}\,.\end{aligned}$$

It follows that for infinitely many $n \in \mathbb{N}$, the algorithm Rec computes f successfully with noticeable probability over $x \sim \mathbf{x}_n$, in time $\bar{T}^{1-\delta/2}$. This contradicts our assumption.

Derandomization in near-linear time with logarithmic advice. Let $c \in \mathbb{N}$. Given input $x \in \{0,1\}^n$, we use the targeted generator as above, but instead of A'' we consider the following

algorithm B. Let $|W(x)|$ be the number of outputs of the generator, and let $\ell = \log(|W(x)|)$.[28] Let

$$\mathsf{Ext}\colon \{0,1\}^{\bar{\ell}} \times [D] \to \{0,1\}^{\ell}$$

be the extractor from Theorem 3.12, instantiated with a sufficiently small constant $\alpha > 0$, and with error $\epsilon = \frac{1}{d(n)\cdot \mathrm{polylog}(T)}$ and min-entropy $k = k(\bar{\ell})$ such that $2^{k-\bar{\ell}} = n^{-\omega(1)}$. Specifically, we can choose $\bar{\ell} = 2\ell \cdot \log(\ell)$ and $k = \ell \cdot \log(\ell)$, in which case $2^{k-\bar{\ell}} = 2^{-\ell\cdot\log(\ell)} < n^{-\omega(1)}$. Note that for these parameters we have $[D] = \mathrm{poly}(\bar{\ell}, d(n), \log(T)) = n^{o(1)}$, and note that $\mathsf{Ext}(z,i)$ (for $z \in \{0,1\}^{\bar{\ell}}, i \in [D]$) is computable in time $\mathrm{poly}(\bar{\ell}) = \mathrm{polylog}(n)$.

We define $B(x,z) = \vee_{i\in[D]} A'(x, W(x)_{\mathsf{Ext}(z,i)})$, where $W(x)_{\mathsf{Ext}(z,i)}$ denotes the v^{th} string in $W(x)$ where v is the integer whose binary representation is $\mathsf{Ext}(z,i)$. Note that B runs in time

$$\bar{T}^{1+O(\sqrt{\delta_{\mathsf{NW}}})} + n^{o(1)} \cdot T^{1+\delta_{\mathsf{NW}}} < T^{1+\epsilon}\,,$$

and uses $\bar{\ell} = \tilde{O}(\log n)$ random coins.

Let $R = \mathsf{Rec}^{D_x}$ be the algorithm defined above, and let us call an input x good if $\Pr[R(x) \neq f(x)] < 2/3$, where the probability is over the random coins of R (i.e., $R(x)$ does not compute $f(x)$). We claim that for every good $x \in \Pi^{\mathsf{Y}}$, the algorithm B accepts x with high probability over z. We prove this statement in two steps:

Claim 5.1.1. *For every good $x \in \Pi^{\mathsf{Y}}$ it holds that $\Pr_{w\in W(x)}[A'(x,w) = 1] \geq \frac{1}{3d'}$, where $d' = d(n) \cdot \mathrm{polylog}(T)$.*

Proof. Let $L^{(1)}, ..., L^{(d')}$ be the d' lists that the generator outputs on x, and observe that $W(x) = \cup_{i\in[d']} L_i$. If for every $i \in [d']$ it holds that $\Pr_{w\in L^{(i)}}[A'(x,w) = 1] < 1/3$, then for every $i \in [d']$ we have that D_x is a $(1/6)$-distinguisher for the uniform distribution on $L^{(i)}$; by Theorem 4.2, in this case $\Pr[R(x) = f(x)] \geq 2/3$, contradicting the goodness of x. Hence, there is $i \in [d']$ such that $\Pr_{w\in L^{(i)}}[A'(x,w) = 1] \geq 1/3$. □

Claim 5.1.2. *For every good $x \in \Pi^{\mathsf{Y}}$ it holds that $\Pr_{z\in\{0,1\}^{\bar{\ell}}}[B(x,z) = 1] \geq 1 - n^{-\omega(1)}$.*

Proof. Let $T_x(u) = A'(x, W(x)_u)$. By Claim 5.1.1, we have that $\Pr_{u\in\{0,1\}^w}[T_x(u) = 1] \geq 1/3d'$. Since we instantiated Ext with error $1/O(d')$, and using Theorem 3.13, with probability at least $1 - n^{-\omega(1)}$ over z there is at least one $i \in [D]$ such that $T_x(\mathsf{Ext}(z,i)) = 1$, in which case $B(z) = 1$. □

Now, recall that for every polynomial-time samplable $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ and every large enough $n \in \mathbb{N}$, with all but negligible probability over $x \sim \mathbf{x}_n$ it holds that x is good. For every input length n, consider the first $t = \log(n)$ Turing machines according to some predetermined enumeration. These t machines induce at most t polynomial-time samplable distributions $\mathbf{x}_n^{(1)}, ..., \mathbf{x}_n^{(t)}$

[28] We assume wlog that $|W(x)|$ is a power of two.

on $\{0,1\}^n$. By a union-bound, we have that

$$\begin{aligned}
&\Pr_{z\in\{0,1\}^{\bar{\ell}},x_1\sim\mathbf{x}_n^{(1)}\ldots x_t\sim\mathbf{x}_n^{(t)}}\left[\exists j\in[t]: x_j\in\Pi^{\mathsf{Y}}\wedge B(x_j,z)=0\right]\\
&\leq t\cdot\max_{j\in[t]}\left\{\Pr_{z,x\sim\mathbf{x}_n^{(j)}}[x\in\Pi^{\mathsf{Y}}\wedge B(x,z)=0]\right\}\\
&\leq t\cdot\max_{j\in[t]}\left\{\Pr_{x\sim\mathbf{x}_n^{(j)}}[x\text{ is not good}]+\max_{x\text{ that is good}}\left\{\Pr_z[x\in\Pi^{\mathsf{Y}}\wedge B(x,z)=0]\right\}\right\}\\
&\leq n^{-\omega(1)}\,.
\end{aligned}$$

Hence, for every input length n there is a fixed z_n such that for all $j\in[t]$ it holds that $\Pr_{x\sim\mathbf{x}_n^{(j)}}[B(x,z_n)=L(x)]\geq 1-n^{-\omega(1)}$. Our algorithm receives this z_n as advice, and given x it outputs $B(x,z_n)$. Indeed, this algorithm runs in time $T^{1+\epsilon}$ and receives $\bar{\ell}=\tilde{O}(\log n)$ advice bits.

The "furthermore" part. Let B be the algorithm above. Recall that for every good $x\in\Pi^{\mathsf{Y}}$ it holds that $\Pr_y[B'(x,y)=1]\geq 1-n^{-\omega(1)}$, and that B runs in time at most $T^{1+\epsilon/3}$.[29]

Consider the algorithm $G^{\mathsf{genlin}}=G_\ell^{\mathsf{genlin}}$ with running time $T'=T^{1+\epsilon/3}$, with $\ell(T')=\bar{\ell}$, and with error $\epsilon/3$; that is, G^{genlin} stretches $\bar{\ell}^{\epsilon/3}$ bits to $\bar{\ell}$ bits in time $T^{(1+\epsilon/3)^2}<T^{1+\epsilon}$ and fools size-$T^{1+\epsilon/3}$ circuits that are samplable in time $\mathrm{poly}(T)$ (see Assumption 3.9). Let $B''(x)=\vee_{s\in\{0,1\}^{\bar{\ell}^{\epsilon/3}}}B'(x,G^{\mathsf{genlin}}(s))$, and observe that B'' runs in time $T^{1+\epsilon}$.

Assume towards a contradiction that there is a polynomial-time samplable $\mathbf{x}=\{\mathbf{x}_n\}$ such that with noticeable probability over $x\sim\mathbf{x}_n$ it holds that

$$\Pr_{s\in\{0,1\}^{\bar{\ell}^{\epsilon/3}}}[B(x,G^{\mathsf{genlin}}(s))=1]<\Pr_{z\in\bar{\ell}}[B'(x,z)=1]-1/10\,.$$

Then, using $\mathbf{x}$ we can sample in polynomial time and with noticeable success probability a $(1/10)$-distinguisher $D_x(r)=B'(x,r)$ for G^{genlin} that runs in time $T^{1+\epsilon/3}$, a contradiction. Hence, for every polynomial-time samplable $\mathbf{x}=\{\mathbf{x}_n\}_{n\in\mathbb{N}}$ it holds that $\Pr_{x\sim\mathbf{x}_n}\left[x\in\Pi^{\mathsf{Y}}\wedge B''(x)=0\right]\leq n^{-\omega(1)}$. ∎

We now restate Theorem 1.2 and prove it. Compared to Theorem 5.1, we now assume that for every polynomial $T(n)$ there is a corresponding hard function f, and deduce derandomization of $prBPTIME[T]$ for every polynomial $T(n)$.

Theorem 5.2 (Theorem 1.2, restated)**.** *For every $\epsilon>0$ there is $\delta=\Theta(\epsilon)$ such that the following holds. Suppose that the near-linear-time PRG assumption is true, and that for every polynomial $T(n)\geq n$ we have the following. For $\bar{T}=T^{1+\delta}$, there is $f\colon\{0,1\}^*\to\{0,1\}^*$ computable by a sufficiently uniform family of circuits of size $\bar{T}(n)$ and depth $d(n)=n^{o(1)}$ such that for every probabilistic algorithm A running in time $\bar{T}^{1-\delta/2}$, and every polynomial-time samplable distribution $\mathbf{x}=\{\mathbf{x}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n\in\mathbb{N}$, we have that $\Pr_{x\sim\mathbf{x}_n}[A(x)=f(x)]\leq n^{-\omega(1)}$. Then,*

$$prBPTIME[T]\subseteq\mathsf{heur}\text{-}prDTIME[T^{1+\epsilon}]/\tilde{O}(\log n)\,.$$

[29] The running time of B ist at most $\bar{T}^{1+O(\sqrt{\delta_{\mathsf{NW}}})}+n^{o(1)}\cdot T^{1+\delta_{\mathsf{NW}}}$. Previously we upper-bounded the latter expression by $T^{1+\epsilon}$, but taking δ_{NW} to be sufficiently small, we can also upper-bound the expression by $T^{1+\epsilon/3}$.

Furthermore, if we assume the generalized near-linear-time PRG (which is stronger than the near-linear-time PRG), then

$$prBPTIME[T] \subseteq \text{heur-}prDTIME[T^{1+\epsilon}] .$$

Proof. By our hypothesis, for every polynomial $T(n)$, the assumption of Theorem 5.1 holds. The proof of Theorem 5.1 constructs a targeted HSG H that gets input x and outputs a set $H(x) \subseteq \{0,1\}^T$ of size $T^{o(1)}$ such that for every T-time probabilistic algorithm A with one sided error, and every polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$, with all but negligible probability over $x \sim \mathbf{x}_n$, the targeted HSG fools $A(x,\cdot)$. (That is, if $\Pr[A(x,\cdot) = 1] \geq 1/2$ then there is an output r of the targeted HSG such that $A(x,r) = 1$.)

Specifically, in the setting of derandomization with $\tilde{O}(\log n)$ bits of advice z_n, we have

$$H(x) = \left\{ G^{\mathsf{lin}}(W(x)_{\mathsf{Ext}(z_n,i)}) \right\}_{i\in[n^{o(1)}]}$$

and in the setting of derandomization without advice we have

$$H(x) = \left\{ G^{\mathsf{lin}}(W(x)_{\mathsf{Ext}(G^{\mathsf{genlin}}(s),i)}) \right\}_{s\in\{0,1\}^{\tilde{\epsilon}\epsilon/3}, i\in[n^{o(1)}]} .$$

The conclusions then follow from Theorem 3.5. ■

5.2 Variations: Time-space tradeoffs, hardness of learning

In Section 5.2.1 we prove Theorem 1.3, and in Section 5.2.2 we prove Theorem 1.4.

5.2.1 Time-space tradeoffs

Similarly to Section 5.1, we first state and prove a derandomization of $prRTIME[T]$ for a fixed polynomial time bound T, and then obtain derandomization of algorithms with two-sided error using Theorem 3.5.

Theorem 5.3 (free lunch derandomization from time-space tradeoffs)**.** *For every $\epsilon > 0$ there is $\delta = \Theta(\epsilon)$ such that the following holds. Suppose that the near-linear-time PRG assumption is true. Let $T(n) \geq n$ be a polynomial, let $\bar{T} = T^{1+\delta}$, and let $f\colon \{0,1\}^* \to \{0,1\}^*$ be computable in time $\bar{T}(n)$. Assume that for every probabilistic algorithm A running in time $\bar{T}^{1+\delta}$ and space $\bar{T}^{1-\delta}$, and every polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n \in \mathbb{N}$, we have that $\Pr_{x\sim\mathbf{x}_n}[A(x) = f(x)] \leq n^{-\omega(1)}$. Then,*

$$prRTIME[T] \subseteq \text{heur-}prDTIME[T^{2+\epsilon}]$$
$$prRTIME[T] \subseteq \text{heur-}prDTIME[T^{1+\epsilon}]/\tilde{O}(\log n) .$$

Furthermore, if we assume the generalized near-linear-time PRG (which is stronger than the near-linear-time PRG), then

$$prRTIME[T] \subseteq \text{heur-}prDTIME[T^{1+\epsilon}] .$$

Proof. Let $\Pi = (\Pi^Y, \Pi^N) \in prRTIME[T]$, let A be a probabilistic time-T algorithm deciding Π, and let $\epsilon > 0$. Let $\delta = \Theta(\epsilon), \delta_{\mathsf{NW}} = \Theta(\delta^2)$, and A' be as in the proof of Theorem 5.1. Since f is computable in time $\bar{T}$, it is also computable by sufficiently uniform circuits of size $\tilde{O}(\bar{T})$. The

derandomization algorithm A'' is identical to the one in Theorem 5.1, using the circuits for f of size (and depth) $\tilde{O}(T)$.

For the analysis, assume that there is a polynomial-time samplable $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ such that for infinitely many $n \in \mathbb{N}$, with noticeable probability over $x \sim \mathbf{x}_n$ it holds that $x \in \Pi^Y$ and $A''(x) = 0$. For every such x, when running Rec from Theorem 4.2 with the distinguisher $D_x(u) = A'(x,u)$, it outputs $f(x)$ with high probability. The running time of Rec is

$$\begin{aligned}(d+n)\cdot \bar{T}^{O(\sqrt{\delta_{\mathsf{NW}}})} + \bar{T}^{O(\sqrt{\delta_{\mathsf{NW}}})}\cdot T^{1+\delta_{\mathsf{NW}}} &< \bar{T}^{1+\delta/4} + \bar{T}^{\delta/4}\cdot T^{1+\delta_{\mathsf{NW}}} && (d=\bar{T})\\ &< \bar{T}^{1+\delta/4}\,,\end{aligned}$$

and the space that it uses is at most $O(n) + \bar{T}^{O(\sqrt{\delta})} + T^{1+\delta_{\mathsf{NW}}} < \bar{T}^{1-\delta/2}$, a contradiction.

The extensions of the conclusion (i.e., to derandomization in near-linear-time with $\tilde{O}(\log n)$ advice, and to derandomization in near-linear-time without advice based on the generalized near-linear-time PRG hypothesis) follow identically to the proof of Theorem 5.1. ■

Theorem 1.3 now follows as a corollary:

Theorem 5.4 (Theorem 1.3, restated)**.** *For every $\epsilon > 0$ there is $\delta = \Theta(\epsilon)$ such that the following holds. Suppose that the near-linear-time PRG assumption is true, and that for every polynomial $T(n) \geq n$ we have the following. For $\bar{T} = T^{1+\delta}$, there is $f\colon \{0,1\}^* \to \{0,1\}^*$ be computable in time $\bar{T}(n)$ such that for every probabilistic algorithm A running in time $\bar{T}^{1+\delta}$ and space $\bar{T}^{1-\delta}$, and every polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n \in \mathbb{N}$, we have that $\Pr_{x\sim\mathbf{x}_n}[A(x) = f(x)] \leq n^{-\omega(1)}$. Then,*

$$pr\mathcal{BPTIME}[T] \subseteq \mathsf{heur}\text{-}pr\mathcal{DTIME}[T^{1+\epsilon}]/\tilde{O}(\log n)\,.$$

Furthermore, if we assume the generalized near-linear-time PRG (which is stronger than the near-linear-time PRG), then

$$pr\mathcal{BPTIME}[T] \subseteq \mathsf{heur}\text{-}pr\mathcal{DTIME}[T^{1+\epsilon}]\,.$$

Proof. The proof is identical to the proof of Theorem 5.2. Specifically, note that the hitting-set generators constructed in the proofs of Theorem 5.1 and Theorem 5.3 are identical, so we can apply Theorem 3.5 in the exact same way as in the proof of Theorem 5.2. ■

5.2.2 Hardness of learning

The following result is a straightforward adaptation of the classical Nisan-Wigderson PRG, instantiated for small output length (see, e.g., [CT21a, Appendix A] and [LP22b, Appendix B]).

Definition 5.5 (learning from membership queries)**.** *Let $f \in \{0,1\}^k$. We say that a probabilistic algorithm A* learns *f* from membership queries with accuracy *$1-\delta$ if, when given input k (in binary) and oracle access to f, with probability at least $2/3$ the algorithm A prints a circuit C such that $\Pr_{i\in[k]}[C(i) = f_i] \geq 1-\delta$. We also extend the notion to settings in which A has an additional oracle D, and the circuit C satisfies $\Pr_{i\in[k]}[C^D(i) = f_i] \geq 1-\delta$; in this case we say A* learns *f* from membership queries with a *D*-oracle. *We sometimes give A an input x (instead of k in binary), in which case we say that A* learns *f* from membership queries with input *x.*

Theorem 5.6. *For every two constants $\epsilon_{\mathsf{NW}}, \delta_{\mathsf{NW}} \in (0,1)$ there is an oracle machine G and a probabilistic oracle machine R such that for every $f \in \{0,1\}^k$:*

- **Generator.** *The machine $G(f)$ runs in time* $\mathrm{poly}(k)$ *and outputs a set of strings in* $\{0,1\}^m$*, where* $m = k^{\epsilon_{\mathsf{NW}}}$.
- **Reconstruction.** *For every* $(1/m)$*-distinguisher* D *for* $G(f)$*, the machine* R *learns* f *from membership queries with a* D*-oracle and accuracy* $1-\delta_{\mathsf{NW}}$ *in time* $\mathrm{poly}(m)$.

The proof of Theorem 1.4 amounts to a straightforward application of Theorem 5.6, recasting the generator G as a targeted generator (i.e., computing $x \mapsto f(x)$ and using $f(x)$ as a truth-table for G; see, e.g., [CT21a; LP22a; LP22b]).

Theorem 5.7 (free lunch derandomization and hardness of learning)**.** *Suppose that the near-linear-time PRG assumption is true. Then, the following two statements are equivalent:*

1. *For every sufficiently small* $\epsilon > 0$ *every* $\delta \in (0,1/2)$ *and every polynomial* $T(n) \geq n$ *there exists* $f\colon \{0,1\}^* \to \{0,1\}^*$ *mapping* n *bits to* $k = n^\epsilon$ *bits that is computable in time* $O(T^{1+4\epsilon})$*, and that for some constant* $\eta > 0$ *satisfies the following. For every probabilistic oracle machine* M*, and every polynomial-time samplable distribution* $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$*, and every sufficiently large* $n \in \mathbb{N}$*, with all but negligible probability over* $x \sim \mathbf{x}_n$*, the machine* M *does not learn* $f(x)$ *from* k^η *membership queries with input* x *and accuracy* $1-\delta$ *in time* $T^{1+\epsilon}$.
2. *For every* $\epsilon > 0$ *and polynomial* $T(n) \geq n$ *it holds that* $prBPTIME[T] \subseteq \mathsf{heur}\text{-}prDTIME[T^{1+\epsilon}]$.

Proof. We first prove that **Item** (1) **implies Item** (2). Let $\Pi \in prBPTIME[T]$, let A be a probabilistic time-T algorithm deciding L, and let $\epsilon > 0$.

Let us first fix appropriate parameters. We instantiate the hypothesis with a sufficiently small $\epsilon' = \Theta(\epsilon)$ and with $\delta = .01$, to obtain a function $f\colon \{0,1\}^n \to \{0,1\}^k$ where $k = n^{\epsilon'}$. Let G be the algorithm from Theorem 5.6, instantiated with $\epsilon_{\mathsf{NW}} = \eta/c$ and $\delta_{\mathsf{NW}} = .01$ where $c > 1$ is a sufficiently large universal constant. For $\delta = \epsilon' \cdot \epsilon_{\mathsf{NW}}$, we instantiate G^{lin} with stretch $n^\delta \mapsto T$ and define A' be as in the proof of Theorem 5.1. Note that A' runs in time $O(T^{1+\epsilon'\cdot\epsilon_{\mathsf{NW}}})$.

- The algorithm: Given input x, we compute $f(x) \in \{0,1\}^k$, compute $R = G(f(x))$ (which is a list of strings of length $m = k^{\epsilon_{\mathsf{NW}}} = n^\delta$), and output $\mathsf{MAJ}_{r\in R}\{A'(x,r)\}$. We denote this algorithm by A'', and note that A'' runs in time $O(T^{1+4\epsilon}) + \mathrm{poly}(k)\cdot T^{1+\delta} < T^{1+\epsilon}$.
- The analysis: Assume that there is a polynomial-time samplable $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ such that for infinitely many $n \in \mathbb{N}$, with noticeable probability over $x \sim \mathbf{x}_n$ it holds that $A''(x) \neq L(x)$. For every such x, the algorithm R learns $f(x)$ from membership queries with oracle $D_x(u) = A'(x,u)$ and accuracy 0.99 in time $\mathrm{poly}(m) = k^{O(\epsilon_{\mathsf{NW}})} < k^\eta$. Simulating R^{D_x} in the obvious way, we obtain an algorithm that learns $f(x)$ with accuracy .99 in time $O(k^\eta \cdot T^{1+\epsilon'\cdot\epsilon_{\mathsf{NW}}}) < T^{1+\epsilon}$, a contradiction.

Let us now show that **Item** (2) **implies Item** (1). Let $\epsilon > 0$ be sufficiently small, let $\delta \in (0,1/2)$, and let $\eta < 1$ (indeed, any choice of $\eta < 1$ suffices). We describe an algorithm that gets $x \in \{0,1\}^n$ and outputs $f(x) \in \{0,1\}^{k=n^\epsilon}$.

We say that $z \in \{0,1\}^k$ is good for x if for every $i \in [\log(n)]$, the i^{th} probabilistic oracle machine equipped with input x does not learn z from k^η membership queries with accuracy $1-\delta$ in time $T^{1+\epsilon}$. Let $\Pi = (\Pi_i^{\mathsf{Y}}, \Pi_i^{\mathsf{N}})$ where

$$\Pi^{\mathsf{Y}} = \left\{ (x \in \{0,1\}^n, i \in \{0,\ldots,k\}, \tau_i \in \{0,1\}^i) : \Pr_{\sigma\in\{0,1\}^{k-i}}[\tau_i \circ \sigma \text{ is good for } x] \geq 2/3 - (i-1)/k - 1/2k \right\},$$

$$\Pi^{\mathsf{N}} = \left\{ (x \in \{0,1\}^n, i \in \{0,\ldots,k\}, \tau_i \in \{0,1\}^i) : \Pr_{\sigma\in\{0,1\}^{k-i}}[\tau_i \circ \sigma \text{ is good for } x] \leq 2/3 - (i-1)/k - 1/k \right\}.$$

Observe that $\Pi \in prBPTIME[\tilde{O}(T^{1+\epsilon} \cdot k)]$, and hence by our assumption $\Pi \in \text{heur-}prDTIME[T^{1+3\epsilon}]$. Let D be the corresponding deterministic time-$T^{1+3\epsilon}$ algorithm for Π.

Let $\tau_0 = \lambda$ be the empty string. For $i = 1, ..., k$, we enter the i^{th} iteration with $\tau_{i-1} \in \{0,1\}^{i-1}$, hoping that there exists $b \in \{0,1\}$ such that $(x, i, \tau_{i-1}b) \in \Pi^{\mathsf{Y}}$. If $D(x, i, \tau_{i-1}0) = 1$ we define $\tau_i = \tau_{i-1}0$, otherwise we define $\tau_i = \tau_{i-1}1$. At the end we output $f(x) = \tau_k$.

The algorithm above runs in time $O(k \cdot T^{1+3\epsilon}) \leq O(T^{1+4\epsilon})$. Turning to the analysis, note that for each $i \in [k]$, if $\tau_{i-1} \notin \Pi^{\mathsf{N}}$, then there exists $b \in \{0,1\}$ such that $(x, i, \tau_{i-1}b) \in \Pi^{\mathsf{Y}}$; this is since $\Pr_\sigma[\tau_{i-1}\sigma \text{ is good for } x] > 2/3 - (i-1)/k - 1/k > 2/3 - i/k - 1/2k$. In particular, assuming that D correctly solves Π on $\{(x, i, \tau_{i-1}b)\}_{b\in\{0,1\}}$, there exists $b \in \{0,1\}$ such that $D(x, i, \tau_{i-1}b) = 1$, and for any such b we have $\tau_{i-1}b \notin \Pi^{\mathsf{N}}$. Also note that $\tau_0 \in \Pi^{\mathsf{Y}}$ (in particular, $\tau_0 \notin \Pi^{\mathsf{N}}$):

Claim 5.7.1. *For any $x \in \{0,1\}^n$ it holds that $(x, 0, \lambda) \in \Pi^{\mathsf{Y}}$.*

Proof. For any fixed $i \in [\log(n)]$, let M_i be the i^{th} probabilistic oracle machine. Consider

$$\Pr_{\sigma\in\{0,1\}^k, r}[M_i(x, r)^\sigma \text{ learns } \sigma \text{ with accuracy } 1 - \delta] \tag{5.1}$$

where r are random coins for M_i.

We first argue that Eq. (5.1) is upper bounded by $2^{-\Omega(k)}$. To do so, fix r to coins that maximize Eq. (5.1), and let M_i' be the corresponding deterministic procedure such that $\Pr_\sigma[M_i'(x)^\sigma \text{ learns } \sigma]$ upper bounds Eq. (5.1).[30] Consider a choice of σ made by first answering the k^η queries of M_i', choosing random values on-the-fly, and then choosing the rest of the $k - k^\eta$ bits of σ. Since the query answers determine an output C of M_i', with high probability the truth-table of C agrees with approximately half of the remaining $k - k^\eta$ entries of σ; specifically, for any $\delta' < 1/2$, with probability at least $1 - 2^{-\Omega((k-k^\eta))} = 1 - 2^{-\Omega(k)}$ it holds that σ disagrees with the truth-table of C on at least $\delta' \cdot (k - k^\eta) = (\delta' - o(1)) \cdot k$ of the k entries. When instantiating the latter fact with $\delta' \in (\delta, 1/2)$, the truth-table of C disagrees with σ on more than $\delta \cdot k$ of the k entries.

By a union-bound over the first $\log(n)$ machines,

$$\Pr_\sigma[\exists i \in [\log(n)] : M_i^\sigma \text{ learns } \sigma \text{ with input } x \text{ and accuracy } 1 - \delta] \leq \log(n) \cdot 2^{-\Omega(k)}$$

which is upper bounded by $1/3$. □

Hence, for every $x \in \{0,1\}^n$, as long as D does not err on the inputs $\{(x, i, \tau_{i-1}b)\}_{b\in\{0,1\}}$ that arise during the execution of the algorithm for f, we have that $f(x)$ is good for x.

The only missing part is to claim that for any polynomial-time samplable $\mathbf{x} = \{\mathbf{x}_n\}$, with all but negligible probability over $x \sim \mathbf{x}_n$ it holds that D does not err on the inputs that arise during the execution of the algorithm. To see this, assume otherwise, and consider an algorithm that samples $x \sim \mathbf{x}_n$, simulates the algorithm for $f(x)$, and uniformly outputs one of the strings $\{(x, i, \tau_{i-1}b)\}_{b\in\{0,1\}}$. With noticeable probability, this algorithm finds an input on which D errs in deciding Π – a contradiction. ■

6 Nondeterministic superfast derandomization

In this section we present our results concerning non-deterministic superfast derandomization. Throughout the section, we will consider $\mathcal{MA}$ protocols with *perfect completeness*.

[30] The subscript M_i' does not indicate that this is the i^{th} machine in the efficient enumeration of TMs, but that M_i' is the procedure obtained by fixing the coins for the i^{th} machine M_i.

Recall that we will be derandomizing $\mathcal{MA}$ protocols into cs-$\mathcal{NP}$ protocols, as defined by Chen and Tell [CT23b] and studied further by Chen, Rothblum, and Tell [CRT25]. Let us recall the definition of cs-$\mathcal{NP}$ protocols, while specifically considering cs-$\mathcal{NP}$ protocols for $\mathcal{MA}$ languages (in which case the honest prover receives a witness in an $\mathcal{MA}$-relation for the language).[31]

Definition 6.1 (computationally sound $\mathcal{NP}$ protocols for $\mathcal{MA}$ languages)**.** *Let $L \in \mathcal{MATIME}[T]$, decided by a T-time verifier V_L. We say that $L \in$ cs-$\mathcal{NTIME}[T_V]$ if there is a uniform deterministic verifier V and a uniform deterministic honest prover P such that:*

1. *(**Completeness.**) For every $x \in \{0,1\}^*$ and every $w \in \{0,1\}^{T(|x|)}$ such that $\Pr[V_L[(x,w) = 1] = 1$ it holds that $V(x, P(x,w)) = 1$.*
2. *(**Computational soundness.**) For every uniform probabilistic adversary $\tilde{P}$ running in time polynomial in T, and every sufficiently large $n \in \mathbb{N}$, the probability that $\tilde{P}(1^n)$ prints (x,π) such that $|x| \in \{0,1\}^n \setminus L$ and $V(x,\pi) = 1$ is negligible in $T(n)$.*
3. *(**Efficiency.**) On input (x,w) the honest prover P runs in deterministic time $\mathrm{poly}(|x| + |w|) = \mathrm{poly}(T(|x|))$. On input (x,π), the verifier runs in deterministic time $T_V(|x|)$.*

In Section 6.1 we prove Theorem 1.5, and in Section 6.2 we prove the extension of the latter theorem that was described in Section 2.2.

6.1 Warm-up: Hardness of $\mathcal{FP}$ for $\mathcal{MA}$ over all polytime distributions

We first present the targeted generator that the proof of Theorem 1.6 will rely on, and then restate the theorem and prove it.

6.1.1 A superfast non-deterministic targeted generator: Basic version

The following construction is the superfast "bare-bones" version of the generator of van Melkebeek and Sdroiveski [MS23] that was described in Section 2.2.

Theorem 6.2 (superfast non-deterministic targeted generator)**.** *Let f be a function mapping n bits to $m = m(n) \leq n$ bits such that f is computable in time T. Then, for every sufficiently small constant $\delta > 0$ there is a deterministic verifier NGen_f and a probabilistic oracle machine NRec_f that satisfy the following:*

1. **Somewhere-PRG.** *When NGen_f gets input $z \in \{0,1\}^n$, it runs in time $\bar{T} = T^{1+O(\sqrt{\delta})}$ and prints a list of M-bit strings, where $M = T^{\delta}$.*
2. **Reconstruction.** *The algorithm NRec_V gets input $z \in \{0,1\}^n$ and oracle access to a function $D\colon \{0,1\}^M \to \{0,1\}$. It runs in time $T^{1-O(\sqrt{\delta})}$, guesses a witness w', tosses random coins r, makes at most $T^{O(\sqrt{\delta})}$ queries to its oracle, and for every $z \in \{0,1\}^n$ satisfies the following.*
   (a) *(**Completeness.**) For every $(1/M)$-distinguisher D for $\mathsf{NGen}_f(z)$ there is w' such that $\Pr_{r'}[\mathsf{NRec}_f^D(z,r,w') = f(z)] = 1$.*
   (b) *(**Soundness.**) For any D and any w' it holds that $\Pr_r[\mathsf{NRec}_f^D(z,r,w') \in \{f(x), \perp\}] \geq 1/2$.*

[31] Recall that supplying the honest prover with a witness (when deciding problems in $\mathcal{NP}$ or in $\mathcal{MA}$) is the standard approach in argument systems, dating back to the 1990s (see, e.g., [Gol01, Section 4.8]).

The construction will rely on two standard technical tools: The Nisan-Wigderson generator, instantiated with small output length; and quasilinear PCPs.

Theorem 6.3 (quasilinear PCP [BSGH+06]). *Let $L \in \mathcal{DTIME}[T]$. Then, there exist two algorithms $P^{\mathsf{PCP}}, V^{\mathsf{PCP}}$ that satisfy the following.*

1. **(Honest prover.)** *When P^{PCP} gets input $x \in L$ it runs in time $\tilde{O}(T)$ and prints a string $P^{\mathsf{PCP}}(x)$.*
2. **(Verifier complexity.)** *On input $x \in \{0,1\}^n$ and when given oracle access to $\pi \in \{0,1\}^{\tilde{O}(T)}$, the verifier V^{PCP} draws random coins $r \in \{0,1\}^{\log(T)+O(\log\log T)}$, runs in time $\tilde{O}(n) + \mathrm{polylog}(T)$, and makes $\mathrm{polylog}(T)$ queries.*
3. **(Completeness.)** *For every $x \in L$, when given oracle access to $P(x) \in \tilde{O}(T)$, the verifier V^{PCP} accepts with probability one.*
4. **(Soundness.)** *For every $x \notin L$ and every $\pi \in \{0,1\}^{\tilde{O}(T)}$, when given input x and oracles access to π, the verifier V^{PCP} accepts with probability at most $1/n$.*

We instantiate Theorem 3.10 with parameters $N = \tilde{O}(T)$ and $\epsilon_{\mathsf{NW}} = \delta$. For simplicity we denote $\mathsf{NGen} = \mathsf{NGen}_f, \mathsf{NRec} = \mathsf{NRec}_f$.

Generator. The algorithm NGen gets input $z \in \{0,1\}^n$ and computes $f(z)$. Let V^{check} be the time-$O(T)$ algorithm that gets $(z,y) \in \{0,1\}^n \times \{0,1\}^m$ and accepts iff $y = f(z)$. Note that $V^{\mathsf{check}}(z, f(z)) = 1$. Let P^{PCP} be the honest prover for V^{check} from Theorem 6.3. The generator NGen computes $w'' = P^{\mathsf{PCP}}(z, f(z))$, and outputs $G_{\mathsf{NW}}(w'')$. Note that NGen outputs strings of length $N^\delta > T^\delta$ (and we can truncate them to length T^δ without any loss), and that the running time of NGen is dominated by the time it takes to run G_{NW}, which is $T^{1+O(\sqrt{\delta})}$.

Reconstruction. On input $z \in \{0,1\}^n$, the reconstruction NRec non-deterministically guesses an output $y' \in \{0,1\}^m$, and non-deterministically guesses advice adv for the algorithm R_{NW} from Theorem 3.10. For $\ell = \log(\tilde{O}(T))$, define $R\colon \{0,1\}^\ell \to \{0,1\}$ by such that $R(i)$ equals the output of R_{NW} with input i and advice adv and oracle access to D. The reconstruction NRec runs the PCP verifier V^{PCP} for V^{check} with input (z,y'), answering its oracle queries with R. If V^{PCP} accepts then NRec outputs y', otherwise it outputs $\perp$.

The running time of NRec is

$$O\left(m + N^{1-O(\sqrt{\delta})} + \mathrm{polylog}(n) \cdot N^{O(\sqrt{\delta})}\right) \leq T^{1-O(\sqrt{\delta})} .$$

Also, since V^{PCP} makes at most $\mathrm{polylog}(T)$ queries to R and R runs in time $T^{O(\sqrt{\delta})}$, the total number of queries that NRec makes is at most $T^{O(\sqrt{\delta})}$.

For completeness, fix $z \in \{0,1\}^n$, and let D be a $1/M$-distinguisher for $\mathsf{NGen}_f(z) = G_{\mathsf{NW}}(w'')$. By Theorem 3.10, there is a string adv such that $R(i) = w''_i$ for all $i \in \{0,1\}^\ell \equiv [|w''|]$. When NRec guesses $y' = y$ and the correct advice adv, the PCP verifier gets oracle access to $r \equiv w''$, and hence has acceptance probability one. In this case NRec outputs $f(z)$ with probability 1.

For soundness, fix $z \in \{0,1\}^n$ and D and $w' = (y', \mathsf{adv})$. Without loss of generality assume that $y' \neq f(x)$, otherwise we are done (as NRec always outputs either its guess y' or $\perp$). Hence, $V^{\mathsf{check}}(z,y') = 0$, and so by the properties of the PCP verifier, for any π, the PCP verifier rejects with probability at least $1 - 1/n$ when given input (z,y') and oracle π. This applies in particular to the witness R defined by adv and R_{NW}, and so NRec_f outputs $\perp$ with high probability.

6.1.2 The derandomization algorithm

We now restate Theorem 1.5 and prove it. The proof follows the outline that was presented in Section 2.2, while obtaining near-linear-time derandomization (rather than quadratic time) using ideas similar to the ones in the proof of Theorem 5.1.

Theorem 6.4 (free lunch derandomization of $\mathcal{MA}$)**.** *For every $\epsilon > 0$ there is $\delta > 0$ such that the following holds. Suppose that the near-linear-time PRG assumption is true. Let $f\colon \{0,1\}^* \to \{0,1\}^*$ mapping n bits to $m = n^{o(1)}$ such that f is computable in deterministic time $T_f(n) = n^{1+O(\sqrt{\delta})}$, and for every $\mathcal{MA}$ verifier V running in time $T_f^{1-\sqrt{\delta}}$, and every polynomial-time samplable distribution $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n \in \mathbb{N}$, we have that $\Pr_{x\sim\mathbf{x}_n}[V(x) \text{ computes } f(x)] \leq n^{-\omega(1)}$.[32] Then, for every polynomial $T(n) \geq n$,*

$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{2+\epsilon}]$$
$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{1+\epsilon}]/\tilde{O}(\log n)\ .$$

Furthermore, if we assume the generalized near-linear-time PRG, then

$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{1+\epsilon}]\ .$$

Proof. Let $\delta = \Theta(\epsilon^2)$. Let $L \in \mathcal{MATIME}[T]$, and let V_L be a T-time $\mathcal{MA}$ verifier for L. We first present the derandomization in near-quadratic time, and later explain how to improve it to near-linear time (with small advice, or relying on the generalized near-linear-time PRG assumption).

Let G^{lin} be the near-linear-time PRG, instantiated with stretch $T^\delta \mapsto T$, and let V'_L be the verifier $V'_L(x,w,r) = V_L(x,w,G^{\mathsf{lin}}(r))$. Note that V'_L uses only T^δ random coins and accepts any (x,w) if and only if V_L accepts (x,w).[33] We will use the generator NGen_f from Theorem 6.2 with parameter $\delta > 0$.

The construction. We now define a derandomized cs-$\mathcal{NTIME}$ verifier V and an honest prover P for L. The verifier gets input x and acts as follows:

- Receives a witness w for V'_L.
- Computes the targeted generator NGen_f from Theorem 6.2 on input $z = (x,w)$ (outputting $\perp$ if NGen_{V_f} rejects). The generator outputs a list $R \subseteq \{0,1\}^M$ of strings of length $M = |z|^\delta > T^\delta$, which we truncate to be of length M.
- If there is $r \in R$ such that $V'_L(x,w,r) = 0$, reject; otherwise, accept.

The prover P gets input x and w such that $\Pr[V_L(x,w) = 1] = 1$, and sends w to the verifier.

Analysis. The running time of P is $O(T(|x|))$, as claimed. For the running time of V, observe that it runs NGen_{V_f} on an input of length $|z| = O(T)$ and with a witness w' of length $T_f(|z|) = T^{1+\delta}$; thus, V runs in time at most

$$T_f(T)^{1+O(\sqrt{\delta})} \cdot T^{1+\delta} < T^{1+\delta+(1+O(\sqrt{\delta}))(1+O(\sqrt{\delta}))} < T^{2+\epsilon}\ .$$

[32] The notion of "$V(x)$ computes $f(x)$" here is the standard one for computing functions in $\mathcal{MA}$: There is w such that $\Pr[V(x,w) = 1] \geq 2/3$, and for every w it holds that $\Pr[V(x,w) \notin \{f(x), \perp\}] \leq 1/3$.

[33] To be fully formal, we can assume wlog that either V_L or V'_L uses naive error-reduction, so that the error of V'_L on each (x,w) is sufficiently small.

For completeness, note that when $x \in L$ and the prover P gets w such that $\Pr[V'_L(x,w) = 1] = 1$, the verifier V gets w and hence outputs $\bigwedge_{r\in R} V'_L(x,w,r) = 1$. As for soundness, assume towards a contradiction that there is a $\mathrm{poly}(T)$-time adversary $\tilde{P}$ such that for infinitely many input lengths $n \in \mathbb{N}$ it holds that

$$\Pr_{(x,w)\leftarrow\tilde{P}(1^n)}[x \in \{0,1\}^n \setminus L \wedge V(x,w) = 1] \geq 1/\mathrm{poly}(T) .$$

For every (x,w) satisfying the above we have that $\Pr_{r\in\mathsf{NGen}_f}[V'_L(x,w,r) = 1] = 1$ whereas $\Pr_{r\in\{0,1\}^M}[V'_L(x,w,r) = 1] \leq 1/2$. Hence, $D_{x,w}(r) = V'_L(x,w,r)$ is a $(1/3)$-distinguisher for $\mathsf{NGen}_f(x,w)$. By the properties of NRec_f from Theorem 6.2 we have that $\mathsf{NRec}_f^{D_{x,w}}(x,w)$ is an $\mathcal{MA}$-verifier that correctly computes $f(x,w)$ in time at most

$$T^{1-O(\sqrt{\delta})} + T^{c\cdot\sqrt{\delta}} \cdot T^{1+\delta} < T^{1+2c\sqrt{\delta}} < T_f^{1-\sqrt{\delta}} , \tag{6.1}$$

where $c > 1$ is a universal constant (from Theorem 6.2) and we relied on a choice of sufficiently large constant $c' > 1$ in the time bound $T_f(n) = n^{1+c'\cdot\sqrt{\delta}}$ and on $\delta = \Theta(\epsilon^2)$ being sufficiently small. This contradicts the hardness of f.

Derandomization in near-linear time with advice. The extension to a near-linear-time algorithm is similar to the one in the proof of Theorem 5.1.

Consider the following modification V' of V. Instead of outputting $\bigwedge_{r\in R} V'_L(x,w,r)$, let

$$\mathsf{Ext}\colon \{0,1\}^{\bar{\ell}} \times [D] \to \{0,1\}^{\ell}$$

be the extractor from Theorem 3.12, instantiated with $\ell = \log(|R|) < (1+\epsilon)\cdot\log(T)$, and with a sufficiently small $\alpha > 0$, error $\epsilon = 1/10$ and min-entropy $k = k(\bar{\ell})$ such that $2^{k-\bar{\ell}} = n^{-\omega(1)}$. Specifically, we choose $\bar{\ell} = 2\ell\cdot\log(\ell)$ and $k = \ell\cdot\log(\ell)$, in which case $D = n^{o(1)}$, and $\mathsf{Ext}(z,i)$ is computable in time $\mathrm{polylog}(n)$. We define V' similarly to V except that it gets random coins $z \in \{0,1\}^{\bar{\ell}}$, and in the last step it outputs

$$V'(x,w,z) = \bigwedge_{i\in[D]} V'_L(x,w,R_{\mathsf{Ext}(z,i)}) .$$

Note that V' runs in time $T_f(T)^{1+O(\sqrt{\delta})} + n^{o(1)}\cdot\tilde{O}(T^{1+\delta}) < T^{1+\epsilon}$, and uses $\bar{\ell} = \tilde{O}(\log n)$ random coins. The proof above shows that for every $\mathrm{poly}(T)$-time $\tilde{P}$, with all but negligible probability it holds that $\tilde{P}(1^n)$ outputs (x,w) such that $D_{x,w}$ is not a $(1/3)$-distinguisher for $R = \mathsf{NGen}_f(x,w)$. We call such (x,w) good, and note that by the properties of Ext, for every good (x,w) we have $\Pr_z[V'(x,w,z) \neq V(x,w)] \leq n^{-\omega(1)}$

For every input length $n+T$, consider the first $t = \log(n)$ Turing machines, which induce at most t polynomial-time samplable distributions $\mathbf{x}_n^{(1)},\ldots,\mathbf{x}_n^{(t)}$ on $\{0,1\}^{n+T}$. By a union-bound,

$$\begin{aligned}
&\Pr_{z\in\{0,1\}^{\bar{\ell}},(x_1,w_1)\sim\mathbf{x}_n^{(1)}\ldots(x_t,w_t)\sim\mathbf{x}_n^{(t)}}\left[\exists j\in[t] : V'(x_j,w_j,z) \neq V(x_j,w_j)\right] \\
&\leq t\cdot\max_{j\in[t]}\left\{\Pr_{z,(x,w)\sim\mathbf{x}_n^{(j)}}[V'(x,w,z) \neq V(x,w)]\right\} \\
&\leq t\cdot\max_{j\in[t]}\left\{\Pr_{(x,w)\sim\mathbf{x}_n^{(j)}}[(x,w) \text{ is not good}] + \max_{(x,w)\text{ that is good}}\left\{\Pr_z[V'(x,w,z) \neq V(x,w)]\right\}\right\} \\
&\leq n^{-\omega(1)} .
\end{aligned}$$

Hence, for every input length n there is a fixed $z_n \in \{0,1\}^{\bar{\ell}}$ such that for all $j \in [t]$ it holds that $\Pr_{(x,w)\sim \mathbf{x}_n^{(j)}}[V'(x,w,z_n) = V(x,w)] \geq 1 - n^{-\omega(1)}$. Our verifier V'' receives this z_n as advice, and given (x,w) it outputs $V'(x,w,z_n)$. Indeed, this verifier runs in time $T^{1+\epsilon}$, receives $\bar{\ell} = \tilde{O}(\log n)$ advice bits, and agrees with V on $1 - n^{-\omega(1)}$ of the inputs, over any polynomial-time samplable distribution.

Derandomization in near-linear time without advice from the generalized near-linear-time PRG assumption. Consider V' from above. We instantiate the generalized near-linear-time PRG with time bound $T^{1+\epsilon}$, with output length $\bar{\ell}$, and with parameter ϵ, to obtain $G = G_{\bar{\ell}}^{\mathsf{genlin}} \colon \{0,1\}^{\bar{\ell}^\epsilon} \to \{0,1\}^{\bar{\ell}}$ computable in time less than $T^{1+3\epsilon}$. Let V'' be a verifier that gets (x,w), behaves identically to V' except that it gets $\bar{\ell}^\epsilon = o(\log(n))$ random coins denoted by z', and it outputs $V'((x,w), G(z'))$. Note that V'' indeed runs in time at most $O(T^{1+3\epsilon})$.

Let V''' be identical to V'' except that it enumerates over the $2^{\bar{\ell}^\epsilon} = T^{o(1)}$ choices for random coins, and V''' runs in time at most $T^{1+4\epsilon}$. We argue that for every polynomial-time samplable distribution $\mathbf{x} = \mathbf{x}_n$ where $\mathbf{x}_n$ is over $\{0,1\}^{n+T}$ and every sufficiently large n it holds that $\Pr_{(x,w)\sim \mathbf{x}_n}[V'''(x,w) \neq V(x,w)] \leq n^{-\omega(1)}$. Indeed, assuming otherwise, $\mathbf{x}$ gives rise to a polynomial-time algorithm S such that $S(1^n)$ finds a circuit $D_{x,w}(z) = V'(x,w,z)$ of size $T^{1+\epsilon}$ satisfying

$$\Pr_{z\in\{0,1\}^{\bar{\ell}}}[D_{x,w}(z) = 1] \notin \Pr_{z\in\{0,1\}^{\bar{\ell}^\epsilon}}[D_{x,z}(G(z)) = 1] \pm .01 \,,$$

which contradicts the assumption. ■

6.2 Hardness of cs-$\mathcal{NP}$ for cs-$\mathcal{MA}$ over all polytime distributions

The goal in this section is to formally state and prove the extension of Theorem 1.5 that was described in Section 2.2. Let us first formally state the result.

We first define a functional version of computationally sound $\mathcal{NP}$. Here we do not supply the honest prover with a witness, because we are not concerned with functions in $\mathcal{NTIME}$ or in $\mathcal{MATIME}$ (and hence the notion of a witness does not exist for these functions). Instead, as in [CT23b; CRT25], we bound the running time of the honest prover.

Definition 6.5 (computationally sound $\mathcal{NP}$)**.** *Let $f\colon \{0,1\}^* \to \{0,1\}^*$ mapping n bits to $m = m(n)$ bits. We say that $f \in$ cs-$\mathcal{NTIME}[T_V, T_P]$ if there is a uniform deterministic verifier V and a uniform deterministic honest prover P such that:*

1. *(**Completeness.**) For every $x \in \{0,1\}^*$ it holds that $V(x, P(x)) = f(x)$.*
2. *(**Computational soundness.**) For every uniform probabilistic adversary $\tilde{P}$ running in time polynomial in T_P, and every sufficiently large $n \in \mathbb{N}$, the probability that $\tilde{P}(1^n)$ prints (x,π) such that $|x| = n$ and $V(x,\pi) \notin \{f(x), \perp\}$ is negligible in $T_P(n)$.*
3. *(**Efficiency.**) On input $x \in \{0,1\}^*$ the honest prover P runs in deterministic time $T_P(|x|)$. On input (x,π), the verifier runs in deterministic time $T_V(|x|)$.*

The next notion refers to cs-$\mathcal{MATIME}$ protocols that compute functions, and these protocols can be defined analogously to Definition 6.5. However, we will be interested in *hardness* for cs-$\mathcal{MATIME}$ protocols, so we need to define exactly what it means for a cs-$\mathcal{MATIME}$ protocol to *fail*. We believe that the definition below is the natural one: We consider an $\mathcal{MA}$ verifier V

that has computational soundness (i.e., soundness is guaranteed against all efficient adversaries), and an honest prover P running in some bounded time, and say that the protocol fails on input x if the honest P does not manage to convince V to output $f(x)$.

Definition 6.6 (hardness for sound cs-$\mathcal{MATIME}$ protocols)**.** *Let $f\colon \{0,1\}^* \to \{0,1\}^*$, let V be a probabilistic verifier running in time T, and let P be a deterministic prover running in time T_P. We say that V and P are a* sound cs-$\mathcal{MATIME}[T, T_P]$ protocol for f *if for every probabilistic $\tilde{P}$ running in time polynomial in T_P, the probability that $\tilde{P}(1^n)$ outputs (x,π) such that $\Pr[V(x,\pi) \notin \{\perp, f(x)\}] \geq 1/2$ is at most $T_P(n)^{-\omega(1)}$. We say that (V,P)* successfully computes $f(x)$ when P gets witness w *if $\Pr[V(x,P(x,w)) = f(x)] \geq 2/3$.*

We are now ready to state the extension of Theorem 1.5. We will assume $f \in$ cs-$\mathcal{NTIME}[T_f, T_P]$ that is hard for sound cs-$\mathcal{MATIME}[T_f^{1-\epsilon}, T_f^{2+\epsilon}]$ over all polynomial-time samplable distributions, where T_f is a near-linear-time function, and deduce free lunch derandomization of $\mathcal{MA}$.

Theorem 6.7 (free lunch derandomization of $\mathcal{MA}$)**.** *For every $\epsilon > 0$ there is $\delta > 0$ such that the following holds. Suppose that the near-linear-time PRG assumption is true. Let $T(n) \geq n$ be a polynomial, let $T_f = n^{1+O(\sqrt{\delta})}$, and let $f\colon \{0,1\}^* \to \{0,1\}^*$ mapping n bits to $m = n^{o(1)}$ such that $f \in$ cs-$\mathcal{NTIME}[T_f, T_P]$. Assume that for every sound* cs-$\mathcal{MATIME}[T_f^{1-\sqrt{\delta}}, T_f^2]$ *protocol (V,P), and every polynomial-time samplable distribution $\mathbf{z} = \{\mathbf{z}_n\}_{n\in\mathbb{N}}$, and every sufficiently large $n \in \mathbb{N}$, the probability over $(x,w) \sim \mathbf{z}_n$ that (V,P) successfully computes $f(x)$ when P gets witness w is at most $T_f(T)^{-\omega(1)}$. Then,*

$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{2+\epsilon}, T_P(T)]$$
$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{1+\epsilon}, T_P(T)]/\tilde{O}(\log n)\,.$$

Furthermore, if we assume the generalized near-linear-time PRG, then

$$\mathcal{MATIME}[T] \subseteq \text{cs-}\mathcal{NTIME}[T^{1+\epsilon}, T_P(T)]\,.$$

In Section 6.2.1 we construct a superfast targeted generator that will be used in the proof of Theorem 6.7, and in Section 6.2.2 we prove the theorem.

6.2.1 A superfast non-deterministic targeted generator: Refined version

We restate the targeted generator from Theorem 2.2 and construct it.

Theorem 6.8 (superfast non-deterministic targeted generator)**.** *Let V be a linear-time algorithm that parses its input as pairs $(z,w') \in \{0,1\}^n \times \{0,1\}^T$, and either rejects or outputs a string of length $m = m(n) \leq n$, and for every z there is w' such that $V(z,w')$ does not reject. Then, for every sufficiently small constant $\delta > 0$ there is a deterministic verifier NGen_V and a probabilistic oracle machine NRec_V that for every $z \in \{0,1\}^N$ satisfy the following:*

1. **Somewhere-PRG.** *When NGen_V gets input $(z,w') \in \{0,1\}^n \times \{0,1\}^T$, it runs in time $\bar{T}$, and either rejects, or prints a list of M-bit strings, where $\bar{T} = T^{1+O(\sqrt{\delta})}$ and $M = T^{\delta}$. The generator NGen_V rejects (z,w') if and only if V rejects (z,w').*
2. **Reconstruction.** *The algorithm NRec_V gets input $z \in \{0,1\}^n$ and oracle access to a function $D\colon \{0,1\}^M \to \{0,1\}$. It runs in time $T^{1-O(\sqrt{\delta})}$, guesses a witness w'', tosses random coins r, makes at most $T^{O(\sqrt{\delta})}$ oracle queries, and satisfies the following.*

(a) (**Efficient honest prover.**) *There is a probabilistic oracle machine* Prv_V *that satisfies the following. For every* (z, w') *such that* $\mathsf{NGen}_V(z, w')$ *does not reject, and every* $(1/M)$*-distinguisher* D *for* $\mathsf{NGen}_V(z, w')$*, the algorithm* $\mathsf{Prv}_V^D(z, w')$ *runs in time* $T^{1+O(\sqrt{\delta})}$ *and with probability at least* $2/3$ *outputs* w'' *such that* $\Pr_{r'}[\mathsf{NRec}_V^D(z, r, w'') = V(z, w')] = 1$.

(b) (**Witnessable soundness.**) *There is a probabilistic polynomial-time oracle machine* Ext' *satisfying the following. For any* D *and any* w''*, if* $\Pr_r[\mathsf{NRec}_V^D(z, r, w'') = y] \geq 1/2$ *for some* $y \in \{0,1\}^*$*, then with probability at least* $2/3$ *the algorithm* $\mathsf{Ext}'(z, w'')$ *outputs* w' *such that* $V(z, w') = y$.

Our targeted generator will rely on the following construction of a PCP that has proofs of near-linear size computable by a prover in near-linear time, and such that convincing PCP witnesses can be "translated back" into proofs for the original verifier. Specifically, we use the construction of Ben-Sasson *et al.* [BSCG+13].[34]

Theorem 6.9 (quasilinear PCP with effective soundness [BSCG+13])**.** *For* $T(n) \geq n$*, let* V *be a linear-time algorithm that parses its input as a pair* $(x, w) \in \{0,1\}^n \times \{0,1\}^{T(n)}$*, and either accepts or rejects. Then, there exist two algorithms* $P^{\mathsf{PCP}}, V^{\mathsf{PCP}}$ *that satisfy the following.*

1. **(Honest prover.)** *When* P^{PCP} *gets input* $(x, w) \in \{0,1\}^n \times \{0,1\}^T$ *such that* $V(x, w)$ *does not reject, it runs in time* $\tilde{O}(T)$ *and prints a string* $P^{\mathsf{PCP}}(x, w) \in \{0,1\}^{\tilde{O}(T)}$.

2. **(Verifier complexity.)** *On input* $x \in \{0,1\}^n$ *and when given oracle access to* $\pi \in \{0,1\}^{\tilde{O}(T)}$*, the verifier* V^{PCP} *draws random coins* $r \in \{0,1\}^{\log(T)+O(\log\log T)}$*, runs in time* $\tilde{O}(n) + \mathrm{polylog}(T)$*, and makes* $\mathrm{polylog}(T)$ *queries.*

3. **(Completeness.)** *On input* $x \in \{0,1\}^n$ *and when given oracle access to* $P(x, w) \in \tilde{O}(T)$ *for some* w *such that* $V(x, w) = 1$*, the verifier* V^{PCP} *accepts with probability one.*

4. **(Proofs of knowledge.)** *There is a probabilistic* $\mathrm{poly}(T)$*-time algorithm* Ext *satisfying the following. Assume that on input* $x \in \{0,1\}^n$ *and when given oracle access to* $\pi \in \{0,1\}^{\tilde{O}(T)}$*, the verifier* V^{PCP} *accepts with probability at least* $1/2$*. Then,* $\mathsf{Ext}(x, \pi)$ *outputs* w *such that* $V(x, w)$ *accepts, with probability at least* $2/3$.

We instantiate the NW generator from Theorem 3.10 with parameters $N = \tilde{O}(T)$ and $\epsilon_{\mathsf{NW}} = \delta$. For simplicity we denote $\mathsf{NGen} = \mathsf{NGen}_f, \mathsf{NRec} = \mathsf{NRec}_f, \mathsf{Prv} = \mathsf{Prv}_f$.

Generator. The algorithm NGen gets input $(z, w') \in \{0,1\}^n \times \{0,1\}^T$, and simulates $V(z, w')$ to obtain a string $y \in \{0,1\}^m$ (if V rejects, then NGen rejects). Let V^{check} be the algorithm that gets $((z, y), w) \in \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^T$, simulates $V(z, w')$, and accepts iff $V(z, w') = y$. Note that $V^{\mathsf{check}}((x, y), w') = 1$. Note that V^{check} runs in time linear in its input length, i.e. $O(n + m + T)$.

Let P^{PCP} be the honest prover for V^{check} from Theorem 6.9. The generator NGen computes $\pi = P^{\mathsf{PCP}}((z, y), w') \in \{0,1\}^{N=\tilde{O}(T)}$, and outputs $G_{\mathsf{NW}}(\pi)$. Note that NGen outputs strings of length $N^\delta > T^\delta$ (and we can truncate them to length T^δ without any loss), and that the running time of NGen is dominated by the time it takes to run G_{NW}, which is $T^{1+O(\sqrt{\delta})}$.

[34] This is the main construction in their paper, and the property of having proofs of knowledge is explicitly stated in [BSCG+13, Definition 12.35 and Claim 12.36].

Reconstruction. On input $z \in \{0,1\}^n$, the reconstruction NRec non-deterministically guesses an output $y' \in \{0,1\}^m$, and non-deterministically guesses advice adv for the algorithm R_{NW} from Theorem 3.10. For $\ell = \log(N)$, define $R\colon \{0,1\}^\ell \to \{0,1\}$ by such that $R(i)$ equals the output of R_{NW} with input i and advice adv and oracle access to D. The reconstruction NRec runs the PCP verifier V^{PCP} for V^{check} with input (z, y'), answering its oracle queries with R. If V^{PCP} accepts then NRec outputs y', otherwise it outputs $\perp$. The running time of NRec on input $z \in \{0,1\}^n$ is

$$O\Big(m + N^{1-O(\sqrt{\delta})} + \mathrm{polylog}(T) \cdot N^{O(\sqrt{\delta})}\Big) \leq T^{1-O(\sqrt{\delta})}$$

and it makes at most $\mathrm{polylog}(T(n)) \cdot T(n)^{O(\sqrt{\delta})} < T^{O(\sqrt{\delta})}$ oracle queries.

For an efficient honest prover, let (z, w') such that $V(z, w') = y$, and let $\pi = P^{\mathsf{PCP}}((z, y), w')$ (where P^{PCP} is the honest prover for V^{check}, as above), so that $\mathsf{NGen}_V(z, w') = G_{\mathsf{NW}}(\pi)$. Fix any $1/M$-distinguisher D for $\mathsf{NGen}_V(z, w') = G_{\mathsf{NW}}(\pi)$. The honest prover gets (z, w') and computes π. It then uses the probabilistic oracle machine P_{NW} from Theorem 3.10. Recall that this machine gets oracle access to D and to the hard truth-table π, runs in time $T^{1+O(\sqrt{\delta})}$, and with high probability outputs advice adv such that R^D_{NW} correctly computes π. The proof that Prv sends to R consists of $w'' = (y, \mathsf{adv})$. Note that V^{PCP} accepts (z, y) with probability 1 when given oracle access to π, and that with high probability $R(i) = \pi_i$ for all $i \in [|\pi|]$. Hence, with high probability Prv sends w'' to NRec so that $\Pr[\mathsf{NRec}^D_V(z, r, w'') = V(z, w')] = 1$. The running time of Prv is dominated by the running time of P_{NW}, and is thus at most $T^{1+O(\sqrt{\delta})}$.

For soundness, fix $z \in \{0,1\}^n$ and assume that for some D and non-deterministic guesses y', adv it holds that NRec_V outputs y' with probability more than $1/2$. Then, V^{PCP} accepts the input (z, y') with probability at least $1/2$ when using oracle R. By the effective soundness of the PCP, when Ext gets input $((z, y'), R)$, with probability at least $2/3$ it outputs w' such that $V^{\mathsf{check}}((z, y'), w') = 1$. Since the witnesses for V^{check} and for V are identical, for every such w' we have that $V(z, w') = y'$. The oracle machine Ext' gets input $(z, (y', \mathsf{adv}))$ and oracle access to D; and it simulates $\mathsf{Ext}((z, y'), \mathsf{adv})$, while giving Ext virtual access to its input R by simulating R_{NW} and answering queries using the oracle D.

6.2.2 Proof of Theorem 6.7

Let $\delta = \Theta(\epsilon^2)$. Let $L \in \mathcal{MATIME}[T]$, and let V_L be a T-time $\mathcal{MA}$ verifier for L. We first explain how to obtain derandomization in near-quadratic time.

As in the proof of Theorem 6.4, let $V'_L(x, w, r) = V_L(x, w, G^{\mathsf{lin}}(r))$ where G^{lin} is the near-linear-time PRG, instantiated with stretch $T^\delta \mapsto T$, and observe that V'_L accepts if and only if V_L accepts. Let V_f and P_f be the cs-$\mathcal{NTIME}[T_f, T_P]$ verifier and prover for f, respectively. We will use the generator NGen_{V_f} from Theorem 6.8 with verifier V_f and parameter $\delta > 0$.

The construction. We now define a derandomized cs-$\mathcal{NTIME}$ verifier V and an honest prover P for L. The verifier gets input x and acts as follows:

- Receives a witness w for V'_L and another witness w' for V_f.
- Computes the targeted generator NGen_{V_f} from Theorem 6.8 on input $z = (x, w)$ and with witness w' (outputting $\perp$ if NGen_{V_f} rejects). The generator outputs a list $R \subseteq \{0,1\}^M$ of strings of length $M = |z|^\delta > T^\delta$, which we truncate to length M.

- If there is $r \in R$ such that $V'_L(x,w,r) = 0$, reject; otherwise, accept.

The prover P gets input x and w such that $\Pr[V_L(x,w)=1]=1$, sends w to the verifier, and simulates P_f on input (x,w) to obtain a witness w' and send it to the verifier.

Analysis. The running time of P is $O(T(|x|)+T_P(|z|)=O(T_P(T))$, as claimed. For the running time of V, observe that it runs NGen_{V_f} on an input of length $|z|=O(T)$ and with a witness w' of length $T_f(|z|)=T^{1+\delta}$; thus, V runs in time at most

$$T_f(T)^{1+O(\sqrt{\delta})} \cdot T^{1+\delta} < T^{1+\delta+(1+O(\sqrt{\delta}))(1+O(\sqrt{\delta}))} < T^{2+\epsilon} \ .$$

Completeness is straightforward: When $x \in L$ and the prover P gets w such that $\Pr[V'_L(x,w)=1]=1$, the prover $P_f(x,w)$ will output a string w' such that $V_f((x,w),w')$ does not reject, and hence NGen_{V_f} does not reject. Since V'_L accepts (x,w) with every random string, it will accept (x,w) with every random string in the output of NGen_{V_f}.

As for soundness, assume towards a contradiction that there is a $\mathrm{poly}(T_P(T))$-time adversary $\tilde{P}$ such that for infinitely many input lengths $n \in \mathbb{N}$ it holds that

$$\Pr_{(x,(w,w'))\leftarrow\tilde{P}(1^n)}\left[x \in \{0,1\}^n \setminus L \wedge V(x,(w,w'))=1\right] \geq 1/\mathrm{poly}(T_P(T(n))) \ .$$

We say that $(x,(w,w'))$ is a violating pair if $x \notin L$ and $V(x,(w,w'))=1$. For every violating pair $(x,(w,w'))$, since V does not reject, it is necessarily the case that NGen_{V_f} did not reject (x,w) with witness w'. Recall that NGen_{V_f} rejects iff V_f rejects, and hence for every violating pair we can define $y(x,(w,w'))=V_f((x,w),w')$. Also, for every violating pair we have

$$V(x,(w,w')) = \bigwedge_{r\in\mathsf{NGen}_{V_f}((x,w),w')} V'_L(x,w,r)=1$$

whereas $\Pr_{r\in\{0,1\}^{T^\delta}}[V'_L(x,w,r)=1] \leq 1/3$, and so $D_{x,w}(r)=V'_L(x,w,r)$ is a $(1/2)$-distinguisher for $\mathsf{NGen}_f(x,(w,w'))$.

Relying on the above, we deduce that for every violating pair $(x,(w,w'))$, the reconstruction NRec_{V_f} from Theorem 6.8 satisfies:

1. The algorithm $\mathsf{Prv}_{V_f}^{D_{x,w}}((x,w),w')$ runs in time $O(T_f(T(|x|)))^2$ and outputs w'' such that $\Pr_{r'}[\mathsf{NRec}_{V_f}^{D_{x,w}}((x,w),r',w'')=y((x,w),w')]=1$.

2. For any w'' and y such that $\Pr_{r'}[\mathsf{NRec}_{V_f}^{D_{x,w}}((x,w),r',w'')=y] \geq 1/2$, the algorithm $\mathsf{Ext}'((x,w),w'')$ outputs, with probability at least $2/3$, a witness w' such that $V_f((x,w),w')=y$.

3. When plugging in $D_{x,w}(r)=V'_L(x,w,r)$, on input $(x,w)\in\{0,1\}^n\times\{0,1\}^T$ the algorithm $\mathsf{NRec}_f^{D_{x,w}}(x,w)$ runs in time

$$T_f(n+T)^{1-O(\sqrt{\delta})} + T_f(n+T)^{c\cdot\sqrt{\delta}} \cdot T(n)^{1+\delta} < T_f(T)^{1-\sqrt{\delta}} \ ,$$

where $c>1$ is a universal constant (from Theorem 6.8) and the calculation is essentially identical to Eq. (6.1), relying on a sufficiently large constant $c'>1$ in the definition of $T_f(T)=T^{1+c'\cdot\sqrt{\delta}}$.

Lemma 6.9.1. *There is a sound* cs-$\mathcal{MATIME}[T_f^{1-\sqrt{\delta}}, T_f^2]$ *protocol* (P, V) *for* f *such that for every violating pair* $(x,(w,w'))$ *satisfying* $V_f((x,w),w') = f(x,w)$*, the protocol successfully computes* $f(x,w)$ *when* P *gets witness* w'*.*

Proof. The verifier V is defined to be $\mathsf{NRec}_{V_f}^{D_{x,w}}$ (i.e., the verifier gets input (x,w), runs NRec_{V_f}, and implements the oracle by $D_{x,w}(r) = V'_L(x,w,r)$). By Item (3) above, the verifier runs in time $T_f(T)^{1-\sqrt{\delta}}$ on input length $n+T$.

The honest prover P uses $\mathsf{Prv}_{V_f}^{D_{x,w}}$. By Item (1) above, when $\mathsf{Prv}_{V_f}^{D_{x,w}}$ is given input (x,w) and witness w', it outputs w'' such that $\Pr[V((x,w),w'') = f(x,w)] = 1$. The running time of $\mathsf{Prv}_{V_f}^{D_{x,w}}$ (while accounting for simulating $D_{x,w}$) is at most $T_f(T)^{1+\epsilon} \cdot T < T_f(T)^2$ on input length $n+T$.

Let us now show that (V,P) is computationally sound, following Definition 6.6. Assume towards a contradiction that there is $\tilde{P}(1^{n+T})$ running in time $\mathrm{poly}(T_f) = \mathrm{poly}(T)$ such that for infinitely many input lengths n, with noticeable probability over $\tilde{P}(1^{n+T}) = ((x,w),w'')$ it holds that $\Pr[V((x,w),w'') \notin \{\bot, f(x)\}] \geq 1/2$. Without loss of generality we can assume that there is y such that $\Pr[V((x,w),w'') = y] \geq 1/2$.[35] By combining $\tilde{P}$ with Ext' from Item (2), we obtain an algorithm running in polynomial time $\mathrm{poly}(T)$ that, with noticeable probability, outputs w' such that $V_f((x,w),w') \neq f(x)$. This contradicts the computational soundness of V_f. □

The proof proceeds by a case-analysis. The first case is that for infinitely many n, with probability at least $1/\mathrm{poly}(T_P(T(n)))$ the algorithm $\tilde{P}(1^n)$ outputs a violating pair $(x,(w,w'))$ such that $V_f((x,w),w') \neq f(x,w)$. In this case we break the computational soundness of V_f. Specifically, consider an algorithm that gets input $1^{n+T(n)}$, simulates $\tilde{P}(1^n)$ to get $(x,(w,w'))$ and outputs $((x,w),w')$; with probability at least $1/\mathrm{poly}(T_P(T(n)))$ this algorithm finds an input (x,w) and proof w' such that $V_f((x,w),w') \neq f(x,w)$, a contradiction.

Hence, we must be in the second case: For infinitely many n, with probability at least $1/\mathrm{poly}(T_P)$ the algorithm $\tilde{P}(1^n)$ outputs a violating pair $(x,(w,w'))$ such that $V_f((x,w),w') = y((x,w),w') = f(x,w)$. In particular, considering the sound protocol from Lemma 6.9.1, this algorithm finds an input (x,w) and auxiliary input w' such that the sound protocol (P,V) successfully computes $f(x,w)$ when P is given witness w'. That is also a contradiction.

The extensions to derandomization in near-linear-time (with advice, or using the generalized near-linear-time PRG) are identical to the proof of Theorem 6.4, and we omit them for brevity.

Remark 6.10. *We explain why hardness assumptions for* cs-$\mathcal{MATIME}$ *arise naturally in this context, and in particular when using a* black-box-$\mathcal{MA}$-reconstructive targeted PRG *to compute* $f \in$ cs-$\mathcal{NP}$*. Let us recall the key problem, as explained in Section 2.2. The efficient algorithm concerning* f *that* V *can use within its time bounds is the* cs-$\mathcal{NP}$ *verifier* V_f*. However, when* V *on input* z *guesses witness for* V_f *it may guess a hard-to-find witness* w *such that* $V_f(z,w) \neq f(z)$*. In this case, on input* z *the verifier* V *does not compute an* $\mathcal{MA}$*-function, since it has several possible outputs.*

The assumption under which the reconstruction procedure works, which asserts that there is an efficient distinguisher D *for the targeted PRG, does not seem helpful in avoiding this problem. To see this, assume that the targeted PRG uses* V_f *to verify the value of* f*, and bases its pseudorandom output list on this value. Then,* D *could be a distinguisher either for a correct value of* f *given by* V_f *or for an incorrect one, or both. In this case, using* D *does not help the reconstruction procedure avoid the issue.*

The point is that the foregoing assumptions on the behavior of these algorithms reflect a broad set of black-box techniques *in constructing PRGs (and targeted PRGs). Specifically, we run into the obstacle*

[35] That is, we can modify V to run multiple times and output $\bot$ unless it always sees the same output. Note that this does not hurt the completeness of V, which is perfect.

above whenever the targeted PRG uses the computational device guaranteed to exist for the hard function (i.e., V_f) to produce pseudorandomness, and whenever the reconstruction relies on this device and on the distinguisher D as a black-box. Targeted PRGs that do not use the distinguisher D as a black-box do exist, but known ones rely either on hardness for conditional time-bounded Kolmogorov complexity [LP22a], or work only for specific weak distinguishers [PRZ23; DPT24; LPT24].

7 Non-batch-computability using downward self-reducibility

In this section, we show that non-batch-computability follows from *worst-case hardness*, for any problem that has two natural properties with a long history of utility:

1. **Downward self-reducibilty:** Solving a single instance of the problem efficiently reduces to solving many smaller instances.

2. **Efficient low-degree extension:** The problem admits an efficient representation as a low-degree multivariate polynomial.

Examples of problems with these properties (and tight hardness conjectures) include Permanent [Val79; CPS99] and Orthogonal Vectors [Wil18].

The worst-case to batch-computability reduction that we show is, essentially, a strong direct product theorem for functions with the properties above. A strong direct product theorem (roughly) says if every time t algorithm fails to correctly evaluate f on a small fraction of inputs, then every time $k \cdot t$ algorithm fails to evaluate the k-wise direct product of f on almost all k-tuples of instances. Recall that strong direct product theorems do not hold for general functions [Sha03], and thus some assumption on the properties of f is necessary.

We will in fact show something even stronger: We (roughly) show that any $k \cdot t$-time algorithm fails to evaluate correctly evaluate f on even a fraction of most k-tuples of instances. This stronger form of non-batch-computability is the one needed in [CT21a], where the key difference is that they assumed such hardness over all polynomial-time samplable distributions, whereas we deduce such hardness over the uniform distribution.

7.1 Preliminaries: Definitions and known results

Let us first recall the notion of non-batch-computability from [CT21a]:

Definition 7.1 (non-batch-computability)**.** *A function $g : \{0,1\}^n \to \{0,1\}^{k(n)}$ is* computable in time T and (ϵ, δ)-non-batch-computable in time $k' \cdot T$ *(with respect to the uniform distribution) if*

1. *There is an algorithm that on input $(x, i) \in \{0,1\}^n \times [k(n)]$ outputs the i^{th} bit of $g(x)$ in time $T(n)$.*

2. *For every probabilistic algorithm A that runs in time $T(n) \cdot k'(n)$, with probability at least δ over $x \xleftarrow{u} \{0,1\}^n$,*

$$\Pr_{i \in [k], A}[A(x)_i \neq g(x)_i] \geq \epsilon$$

To parse the definition, observe that no function $g\colon \{0,1\}^n \to \{0,1\}^k$ computable in time T can be (01.01)-non-batch-computable in time $k \cdot T$ (i.e., $k' = k$ is too much to hope for), since we can trivially compute all k output bits of g in time $k \cdot T$. We will be interested in values of, say, $k' = k^{.01}$.

Downward self-reducibility. Loosely speaking, a problem is downward self-reducible if solving a single instance efficiently reduces to solving many smaller instances. We use a more general definition from Ball et al. [BRS+18], which allows reducing one function $\mathcal{F}$ to another function $\mathcal{F}'$, where we think of $\mathcal{F}'$ as functions over smaller instances.

Definition 7.2 (downward reducibility). *A function family $\mathcal{F} = \{f_n : \mathcal{X}_n \to \mathcal{Y}_n\}$ is (s, ℓ)-*downward reducible in time *t to a function family $\mathcal{F}' = \{f'_n : \mathcal{X}'_n \to \mathcal{Y}'_n\}$ if there is a pair of algorithms* (Divide, Assemble) *satisfying:*

- *For all large enough n, $s(n) < n$.*
- Divide *on input $x \in \mathcal{X}_n$ outputs $\ell(n)$ instances from $\mathcal{X}'_{s(n)}$; that is,*

$$\text{Divide}(x) = (x_1, \ldots, x_{\ell(n)})$$

- *Given the value of $\mathcal{F}'$ at these $\ell(n)$ instances,* Assemble *computes the value of $\mathcal{F}$ at x; that is,*

$$\text{Assemble}(x, f'_{s(n)}(x_1), \ldots, f'_{s(n)}(x_{\ell(n)})) = f_n(x)$$

- *The combined running time of* Divide *and* Assemble *on input $x \in \{0,1\}^n$ is at most $t(n)$.*

If $\mathcal{F}' = \mathcal{F}$, we say that $\mathcal{F}$ is downward self-reducible.

Efficient low-degree extension. We next define what it means for a decision problem to admit an efficient low-degree extension. Intuitively, a Boolean function f admits a (t, d)-low-degree extension if there is a degree d polynomial P over a finite field that can be computed in time t and effectively agrees with f on inputs from the Boolean hypercube; specifically, we require that $P(x) = 0$ when $f(x) = 0$, and $P(x) \neq 0$ otherwise.

In the following statement, we think of the input as of length m. This foresees the application of low-degree extensions on inputs of length $m = m(n)$ that are obtained by applying downward self-reducibility on inputs of length n.

Definition 7.3 (efficient low-degree extensions). *A family of functions $\{f : \{0,1\}^m \to \{0,1\}\}$ admits a (t_1, t_2, d)-*low-degree extension *if there exists a pair of algorithms A, B such that the following holds:*

- *On input 1^m, the algorithm A outputs a representation of a field of size $q = q(m) \leq 2^{m^{o(1)}}$ in time $t_1(m)$, where $q(m)$ is increasing. The remaining items hold with respect to $q = q(m)$.*
- *On input $x \in \mathbb{F}_q^m$, the algorithm $B(x)$ runs in time $t_2(m)$ and computes a polynomial of degree $d = d(m)$. (That is, there exists $P \colon \mathbb{F}_q^m \to \mathbb{F}_q$ of degree d such that $B(x) = P(x)$ for all $x \in \mathbb{F}_q^m$.)*
- *For every $x \in \{0,1\}^m \subseteq \mathbb{F}_q^m$ it holds that $B(x) = 0 \iff f(x) = 0$.*

An example: orthogonal vectors. Ball et al. [BRS+17; BRS+18] observed that the k-Orthogonal-Vectors problem of dimension $(\frac{k}{k+c})^2 d$ is downward reducible to k-Orthogonal-Vectors of dimension d, which in turn reduces to evaluating a family of degree kd polynomials (that can themselves be evaluated in time $\tilde{O}(n^k d)$.

Proposition 7.4 ([BRS+17; BRS+18]). *For any constant integer k and any constant $c > 0$, the problem k-Orthogonal Vectors of size n and dimension $(\frac{k}{k+c})^2 d$ has the following properties:*

1. *It is (m, m^c)-downward reducible, where $m = n^{k/(c+k)}$, in time $\tilde{O}(m^{c+1})$.*

2. *It admits a $(\tilde{O}(n^{k/2}), \tilde{O}(n^k d), kd)$-low-degree extension.*

The randomized Orthogonal Vectors conjecture asserts that k-orthogonal vectors requires n^k probabilistic time to solve when $d = \omega(\log n)$. This conjecture follows from the randomized strong exponential time hypothesis [IPZ01; Wil18], which asserts that for any constant $\epsilon > 0$, there exists a k such that k-SAT requires time $\Omega(2^{(1-\epsilon)n})$. See [Wil18] for further details.

7.2 Weak non-batch-computability from worst-case hardness

We are now ready to state our main theorem for this section, which gives a weak form of non-batchability: any fast algorithm fails to approximately compute the batch with probability bounded away from zero. In the next subsection, we will see how to amplify this result such that any fast algorithm fails to approximately compute almost every batch.

Theorem 7.5 (weak non-batch-computability from worst-case hardness)**.** *Let $\gamma > \gamma' \in (0,1)$ be sufficiently small, and let $\alpha, \beta > 0$. Assume that there exist $f\colon \{0,1\}^* \to \{0,1\}$ and $t(n) \geq n$ such that:*

1. **(Upper bound.)** *f admits a $(t(n)^{1-\alpha}, t(n), n^{o(1)})$-low-degree extension.*[36]

2. **(Lower bound.)** *f is hard for randomized time $O(t(n)^{1-\alpha})$ (in the worst-case) on all input lengths.*

3. **(Downward self-reducibility.)** *f is $(m, n^{\gamma'})$-downward self reducible in time $t(n)^{1-\alpha}$, where $m = m(n)$ is such that $t(m) \cdot n^{3\beta} \leq t(n)^{1-\alpha}$.*

Then, there is a direct product function g mapping $m^{1+o(1)} \cdot n^{\gamma}$ bits to n^{γ} bits such that g is computable in time $O(t(m))$ and (ϵ, δ)-non-batch-computable in time $O(n^{\beta} \cdot t(m))$, where $\epsilon, \delta > 0$ are small universal constants.

The proof of this theorem is in two steps. First, the main lemma, Lemma 7.7, effectively shows that batch polynomial evaluation is no harder than approximate batch polynomial evaluation. This immediately gives non-batch-computability over a large alphabet. We then apply the Goldreich-Levin predicate to reduce to a binary alphabet. (The analysis of this latter transformation only works in the low-error regime, and we show how to generically boost this result to the high-error regime in the next subsection.)

The main lemma: Batch polynomial evaluation from approximate batch polynomial evaluation. The following is the main lemma underlying the proof of Theorem 7.5. Before stating it, we recall the standard Berlekamp-Welch unique decoding algorithm for the Reed-Solomon code.

Theorem 7.6 (unique decoding of RS [WB86]; see, e.g., [AB09, Theorem 19.15])**.** *There is a polynomial-time algorithm that gets as input a list $(a_1, b_1), \ldots, (a_m, b_m)$ of pairs of elements of a finite field $\mathbb{F}$ and satisfies the following. For any $d < m$, if there is a degree-d polynomial $P\colon \mathbb{F} \to \mathbb{F}$ such that $\Pr_{i \in [m]}[P(a_i) = b_i] \geq 1/2 + d/2m$, then the algorithm outputs P.*

Lemma 7.7. *Let $d = d(n), m = m(n), m' = m(n)$ such that $m > 2(m'-1)d$. Let $\epsilon = \epsilon(n), \delta = \delta(n) > 0$ such that $\epsilon + 2\delta < 1/16$.*

[36] Note that this implies that f is computable in time t.

Assume that there exists a randomized algorithm $\mathcal{A}$ that on an input tuple in $(\mathbb{F}^n)^m$ runs in time $T(n,m)$, and for some polynomial $p : \mathbb{F}^n \to \mathbb{F}$ of degree d satisfies

$$\Pr_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m \xleftarrow{u} (\mathbb{F}^n)^m} \left[\Pr_{\mathcal{A}, i \in [m]} [\mathcal{A}(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)_i = p(\boldsymbol{x}_i)] \geq 1 - \epsilon \right] \geq 1 - \delta \,,$$

where $|\mathbb{F}| \geq \max\{4m'd, m\}$.

Then, for any $\delta' = \delta'(n) > 0$, there exists a randomized algorithm $\mathcal{B}$ that gets input $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{m'} \in (\mathbb{F}^n)^{m'}$, runs in time $\mathrm{poly}(m/(\delta\cdot\delta'), \log|\mathbb{F}|) + O(d/\delta\cdot\delta')\cdot T(n,m)$, *and satisfies*

$$\Pr_{\mathcal{B}} [\mathcal{B}(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{m'}) = (p(\boldsymbol{x}_1),\ldots,p(\boldsymbol{x}_{m'})] \geq 1 - \delta' \,.$$

A similar result was recently shown in Ball et al. [BGH+24], but for a different parameter regime (arbitrarily large polynomial size batches and subconstant error/approximation tolerance). We provide a new and more straightforward reduction in the parameter regime of interest here. In addition to its simplicity, one advantage of this direct reduction over the recursive one presented in Ball et al. [BGH+24] is that it does not rely on approximate batch solvers that work for a range of parameters.

Proof of Lemma 7.7. Let $\mathcal{A}$ be as in our hypothesis. Let $t \stackrel{\text{def}}{=\!=} 4d/(\delta\cdot\delta')$. Let $\mathcal{B}$ behave as follows on input $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{m'} \in \mathbb{F}^{n\times m'}$:

1. Let $h : \mathbb{F} \to \mathbb{F}^n$ be the degree $(m'-1)$ curve such that $h(i) = \boldsymbol{x}_i$ for all $i \in [m']$. Let $\boldsymbol{y} = (\boldsymbol{y}_1,\ldots,\boldsymbol{y}_m) = (h(1),\ldots,h(m))$.

   (Note: $q(i) \stackrel{\text{def}}{=\!=} p(h(i))$ is a degree $D = d(m'-1)$ polynomial such that $q(i) = p(\boldsymbol{x}_i)$ for $i \in [m']$.)

2. Now sample uniformly random $\boldsymbol{r} = (\boldsymbol{r}_1,\ldots,\boldsymbol{r}_m), \boldsymbol{s} = (\boldsymbol{s}_1,\ldots,\boldsymbol{s}_m) \in \mathbb{F}^{n\times m}$. For $j = 1,\ldots,t$, define $(\boldsymbol{y}_{1,j},\ldots,\boldsymbol{y}_{m,j}) = \boldsymbol{y} + j\cdot\boldsymbol{r} + j^2\cdot\boldsymbol{s}$.

   (Note (a): For any $i \in [m]$, $q_i(j) = p(\boldsymbol{y}_i + j\cdot\boldsymbol{r}_i + j^2\cdot\boldsymbol{r}_i)$ is a degree $2d$ univariate polynomial such that $q_i(0) = p(\boldsymbol{y}_i)$.)

   (Note (b): For any $j \neq j' \in [t]$, the marginal distribution of $(\boldsymbol{y}_{1,j},\ldots,\boldsymbol{y}_{m,j}), (\boldsymbol{y}_{1,j'},\ldots,\boldsymbol{y}_{m,j'})$ is uniformly random.)

3. For each $j \in [t]$, compute $(z_{1,j},\ldots,z_{m,j}) \leftarrow \mathcal{A}(\boldsymbol{y}_{1,j},\ldots,\boldsymbol{y}_{m,j})$.

4. For each $i \in [m]$, run the algorithm from Theorem 7.6 on input $(1,z_{i,1}),\ldots,(t,z_{i,t})$ to get a polynomial $\hat{q}_i$ of degree $2d$. Let $z_i = \hat{q}_i(0)$.

5. Run the Reed-Solomon decoding algorithm from Theorem 7.6 on input $(1,z_1),\ldots,(m,z_m)$ to get a degree $D = d(m'-1)$ polynomial $\hat{q}$. Output $(\hat{q}(1),\ldots,\hat{q}(m'))$.

Complexity. Notice that $\mathcal{B}$ uses t invocations of $\mathcal{A}$ on inputs in $(\mathbb{F}^n)^m$, and m invocations of the algorithm from Theorem 7.6 on tuples of length $t = O(d/(\delta\cdot\delta'))$, and one additional invocation of the latter algorithm on a tuple of length m. Thus the total complexity of $\mathcal{B}$ is $\mathrm{poly}(m/(\delta\cdot\delta'), \log|\mathbb{F}|) + O(d/(\delta\cdot\delta'))\cdot T(n,m)$.

Correctness. By our observation below Item (1) above, it suffices to show that the polynomial $\hat{q}$ obtained in the last step is precisely q, with probability at least $1-\delta'$. By the correctness of Theorem 7.6 and the observation below Item (2), it suffices to show that with probability at least $1-\delta'$, for a $1/2 + D/2m$ fraction of the indices $i \in [m]$, the degree-d polynomial $\hat{q}_i$ obtained in the penultimate step is precisely q_i.

In turn, to show this, it suffices to show that with probability at least $1-\delta'$, for at least $(m+D)/2$ indices $i \in [m]$ we have

$$\Pr_{j\in[t]}\left[z_{i,j} = p(\boldsymbol{y}_{i,j})\right] \geq 1/2 + (2d)/(2t)\,.$$

Finally, to prove the statement above, it suffices to prove that $z_{i,j} \neq p(\boldsymbol{y}_{i,j})$ for at most $\frac{m-D}{2}\cdot\frac{t-2d}{2}$ pairs $(i,j) \in [m]\times[t]$. By our choice of $D < m/2$ and $2d < t/2$, it suffices to show that $z_{i,j} \neq p(\boldsymbol{y}_{i,j})$ for at most $\frac{mt}{16}$ pairs (i,j).

Now, for each $j \in [t]$, let E_j denote the event that $|\{i : z_{i,j} \neq p(\boldsymbol{y}_{i,j})| > \epsilon m$. Because for each $j \in [t]$ the marginal distribution of $(\boldsymbol{y}_{1,j},\ldots,\boldsymbol{y}_{m,j})$ is uniformly random, we have that $\Pr[E_j] \leq \delta$. Moreover, because $(\boldsymbol{y}_{1,j'},\ldots,\boldsymbol{y}_{m,j})$ and $(\boldsymbol{y}_{1,j''},\ldots,\boldsymbol{y}_{m,j'})$ are independent for any $j' \neq j$, it follows that E_j and $E_{j'}$ are independent. By Chebyshev's inequality, $\Pr[\sum_{j\in[t]} E_j > 2\delta t] \leq \frac{\delta-\delta^2}{\delta^2 t} < 1/\delta t$.

Finally, notice that if $\sum_{j\in[t]} E_j \leq 2\delta t$, then the total number of (i,j) such that $z_{i,j} \neq p(\boldsymbol{y}_{i,j})$ is strictly less than $\epsilon m t + 2\delta t m = mt(\epsilon+2\delta)$. By our assumption that $\epsilon + 2\delta < 1/16$, with probability at most $1/\delta t$, the number of pairs (i,j) such that $z_{i,j} \neq p(\boldsymbol{y}_{i,j})$ is at most $\frac{mt}{16}$. Hence, the algorithm $\mathcal{B}$ is correct with probability at least $1 - 1/\delta t > 1-\delta'$. ∎

Proof of Theorem 7.5. Now that we proved Lemma 7.7, for the rest of this section we prove Theorem 7.5.

Let $\epsilon', \delta' > 0$ be constants such that $\epsilon' + 2\delta' < 1/16$. Let $\epsilon, \delta > 0$ be sufficiently small such that $\delta/\delta' + \epsilon < \epsilon'/8$. (Note that, for example, $\epsilon = \delta = 2^{-16}$ and $\epsilon' = \delta' = 1/64$ suffice.)

Let (A, B) be the algorithms for the low-degree extension of f, let $\ell' = \ell'(n) = n^\gamma$, and let $q = q(m)$ be the size of the field that A produces on inputs of length m. Let $g' = \left\{g'_m \colon \mathbb{F}_q^{m\times\ell'} \to \mathbb{F}_q^{\ell'}\right\}$ be the partial function represented by ℓ'-fold direct product of the low-degree extension of f on inputs with m bits; that is, $g'(x_1,\ldots,x_{\ell'}) = B(x_1),\ldots,B(x_{\ell'})$.

Claim 7.7.1. *There is no randomized algorithm $\mathcal{A}$ that gets input $(x_1,\ldots,x_{\ell'}) \in \mathbb{F}_q^{m\times\ell'}$, where $\ell' = \ell'(n) = n^\gamma$, runs in time $t(m)\cdot n^{2\beta}$, and with probability more than δ' over choice of $(x_1,\ldots,x_{\ell'})$ satisfies*

$$\Pr_{\mathcal{A}}\left[\Pr_{i\in[\ell']}\left[\mathcal{A}(x_1,\ldots,x_{\ell'})_i = g'(x_1,\ldots,x_{\ell'})_i\right] \geq 1-\epsilon'\right] \geq 2/3\,,$$

where the outer probability $\Pr_{\mathcal{A}}$ is over the internal randomness of $\mathcal{A}$.

Proof. Assume towards a contradiction that there exists such an algorithm $\mathcal{A}$. By Lemma 7.7, instantiated with input length m and parameters $d = d(m) = \deg(B)$ and $m = \ell'$ and $m' = \ell$ and $\delta' = 1/3$ and $\epsilon = \epsilon'$ and $\delta = \delta'$, there is an algorithm $\mathcal{B}$ that, for every input $x_1,\ldots,x_\ell$, successfully computes B with probability at least $2/3$ in time

$$\mathrm{poly}(\ell', \log(q)) + O(d)\cdot t(m)\cdot n^{2\beta}\,.$$

(To invoke Lemma 7.7 with these parameters we need to ensure that $\epsilon' + 2\delta' < 1/16$ and that $\ell' > 2\ell d$, which hold by our hypotheses that $d = d(n) = n^{o(1)}$ and $\gamma > \gamma'$.)

Now consider the algorithm F for f, that on input x does the following:

1. Run $\text{Divide}(x) \to (y_1, \ldots, y_\ell)$ where $\ell = n^{\gamma'}$ and $|y_i| = m$.
2. Run $A(m)$ to get a representation of $\mathbb{F}_q$ where $q = q(m) \leq 2^{m^{o(1)}}$ and encode y_i as $\hat{y}_i \in \mathbb{F}_q^m$, for all $i \in [\ell]$. Note that $\hat{y}_i$ is of length $m' = m^{1+o(1)}$.
3. Run $\mathcal{B}(\hat{y}_1, \ldots, \hat{y}_\ell) \to \hat{z}_1, \ldots, \hat{z}_k$.
4. For $i \in [\ell]$, set $z_i = 0$ if $\hat{z}_i = 0$ and set $z_i = 1$ otherwise.
5. Output $\text{Assemble}(z_1, \ldots, z_\ell)$.

By the correctness of $\mathcal{B}$ and $(\text{Divide}, \text{Assemble})$, the algorithm F succeeds in computing f with probability $2/3$. The runtime of F is at most

$$O(t(n)^{1-\alpha}) + \text{poly}(\ell', \log(q)) + O(d) \cdot t(m) \cdot n^{2\beta} ,$$

which is at most $O(t(n)^{1-\alpha})$ by our hypotheses that $t(m) \cdot n^{3\beta} \leq t(n)^{1-\alpha}$ and that $\gamma > 0$ is sufficiently small (this ensures that $\text{poly}(n^\gamma) < n^{1-\alpha} \leq t(n)^{1-\alpha}$, where the "poly" notation refers to a universal polynomial). □

The non-batch-computable function g is the Goldreich-Levin predicate applied to each output symbol of g'; that is, $g(r_1, x_1, \ldots, r_{\ell'}, x_{\ell'}) = (\langle r_1, B(x_1)\rangle, \ldots, \langle r_{\ell'}, B(x_\ell)\rangle)$ where each $x_i \in \{0,1\}^{m \cdot \log(q)} \equiv \mathbb{F}_q^m$, and $\ell' = n^\gamma$, and $\langle x, y\rangle$ is the inner product modulo 2. Note that:

1. The function g maps $\ell' \cdot (m+1) \cdot \log(q) = m^{1+o(1)} \cdot n^\gamma$ bits to $\ell' = n^\gamma$ bits.
2. Each output bit of g is computable in time $O(t(m) + \log(q)) = O(t(m))$, since B is computable in time $t(m) \geq m$.

We now claim that if g is batch-computable, then g' is batch-computable, which contradicts Claim 7.7.1. Let G be a probabilistic algorithm that violates non-batch-computability of g; that is, G runs in time $t(m) \cdot n^\beta$ on inputs of length $m^{1+o(1)} \cdot n^\gamma$, and satisfies

$$\Pr_{r_1,\ldots,r_\ell,x_1,\ldots,x_\ell} \left[\Pr_{i \in [\ell], G} [G(r_1, x_1, \ldots, r_\ell, x_\ell)_i \neq \langle r_i, B(x_i)\rangle\}] \geq \epsilon \right] < \delta . \tag{7.1}$$

To show our claim, we first make two observations.

Claim 7.7.2. *At least $1 - \delta'$ of the tuples $x_1, \ldots, x_\ell \in \{0,1\}^{m \cdot \log(q)}$ are* good*, in the sense that*

$$\Pr_{r_1,\ldots,r_\ell} \left[\Pr_{G, i \in [\ell]} [G(r_1, x_1, \ldots, r_\ell, x_\ell)_i \neq \langle r_i, B(x_i)\rangle\}| \geq \epsilon] \right] < \delta/\delta'.$$

Proof. Let X be the random variable that chooses $x_1, \ldots, x_\ell$ and outputs the LHS of the inequality above. By Eq. (7.1), we have $\mathbb{E}[X] < \delta$, and thus by Markov's inequality $\Pr[X \geq \delta/\delta'] \leq \delta'$. □

Claim 7.7.3. *We say that $i \in [\ell]$ is* good for $x = (x_1, \ldots, x_\ell)$ *if*

$$\Pr_{G, r_1,\ldots,r_\ell} [G(r_1, x_1, \ldots, r_\ell, x_\ell)_i \neq \langle r_i, B(x_i)\rangle] < 1/8 .$$

Then, for every good x, at least $1 - 8(\delta/\delta' + \epsilon)$ of the indices $i \in [\ell]$ are good.

Proof. Let Y be the random variable that chooses a random i and outputs the LHS of the inequality above. Since x is good we have $\mathbb{E}[Y] \leq \delta/\delta' + \epsilon$, and thus by Markov's inequality, $\Pr[Y \geq 1/8] < \frac{\delta/\delta'+\epsilon}{1/8}$. □

Now, to batch-compute g', we run the naive Hadamard local decoder on each inner product. Specifically, given $x_1, ..., x_\ell \in \mathbb{F}_q^m$, we sample $r^{(1)}, ..., r^{(a)}$ uniformly at random, where each $r^{(j)} = (r_1^{(j)}, ..., r_\ell^{(j)}) \in (\{0,1\}^{\log(q)})^\ell$ and a will be determined later, and we compute

$$\begin{aligned} z^{(j)} &= G(r_1^{(j)}, x_1, \ldots, r_\ell^{(j)}, x_\ell) \,, \\ z^{(j,k)} &= G(r_1^{(j)} \oplus e_k, x_1, \ldots, r_\ell^{(j)} \oplus e_k, x_\ell) \quad \forall k \in [\log(q)] \,, \end{aligned}$$

where e_k is the k^{th} standard basis vector in $\{0,1\}^{\log(q)}$. For $i \in [\ell]$ and $j \in [a]$ and $k \in [\log(q)]$, let $b_{i,j,k} = z_i^{(j)} \oplus z_i^{(j,k)}$. For each $i \in [\ell]$ and $k \in [\log(q)]$, let $y_{i,k} = \mathsf{MAJ}_{j \in [a]} \{b_{i,j,k}\}$, and output the string $(y_{1,1}, ..., y_{1,k}), ..., (y_{\ell,1}, ..., y_{\ell,k})$, where each k bits are parsed as a symbol in $\mathbb{F}_q$.

Claim 7.7.4. *Fix any good x, and any $i \in [\ell]$ that is good for x. Then, with probability at least $1 - 2^{-\Omega(a)} \cdot \log(q)$ it holds that $(y_{i,1}, ..., y_{i,k}) = B(x_i)$.*

Proof. Fix any $k \in [\log(q)]$. With probability at least $3/4$ over choice of $r_i^{(j)}$ it holds that $z_i^{(j)} = \left\langle r_i^{(j)}, B(x_i) \right\rangle$ and $z_i^{(j,k)} = \left\langle (r_i^{(j)} \oplus e_k)_i, B(x_i) \right\rangle$, in which case $b_{i,j,k} = B(x_i)_k$. Thus, with probability at least $1 - 2^{-\Omega(a)}$ it holds that $y_{i,j,k} = B(x_i)_k$. The claim follows by a union-bound over $k \in [\log(q)]$. □

Letting $a = O(\log(\ell \cdot \log(q)/\delta))$, on any good x, with probability at least $1 - \delta$, for every good i the i^{th} output of the algorithm above is $B(x_i)$. In particular, with probability at least $1 - \delta'$ over x, the algorithm computes g' correctly on $1 - 8(\delta/\delta' + \epsilon) > 1 - \epsilon'$ of the indices.

Finally, the algorithm for batch-computing g' runs in time

$$O(a \cdot \log(q) \cdot t(m) \cdot n^\beta) \leq \tilde{O}(t(m) \cdot n^\beta) < t(m) \cdot n^{2\beta} \,,$$

a contradiction.

7.3 Approximate direct product amplification

We show how to amplify the success probability of a (approximate) batch solver. This ultimately allows us to generically amplify the result from the last section to show the function is hard for a fast (approximate) batch solver to be correct with even small probability. The only property of the hard function we are using is that it does not take to long to evaluate on a single instance, giving us the ability to locally check a few evaluations (in the non-uniform setting, this can likely be avoided using advice following the techniques of [IJK+10]).

We adopt a clever proof due to Impagliazzo, Jaiswal, and Kabanets [IJK07] for proving chernoff-type direct products in our setting. [IJK07] essentially showed that if a function f is hard to compute correctly with say constant probability, then any algorithm running in essentially the same time fails to compute f correctly on even a fraction of k independent instances with very small probability. We, on the other hand, show that if one can quickly compute f on a fraction of a large batch of instances, then one can efficiently compute f on a similar fraction of a slightly smaller batch of instances with high probability.

Impagliazzo et al.'s reduction works by repeatedly embedding the input instance in a random batch of instances, running the efficient batch solver, testing if the solver is sufficiently correct on the generated batch, and outputting the solution to input instance if the test passes. Their proof works by considering a hybrid experiment where the input is simply sampled from the set of tuples on which the batch solver is successful.

Our reduction follows the same template, except that instead of embedding a single instance, we embed a smaller batch of instances (instead of just one). Modulo a generalization of the key lemma and the use of a concentration bound on hypergeometric distributions, the proof is very similar.

The key lemma in our approximate direct product amplification is a generalization of [IJK07]'s "sampling lemma" to our batched setting.

Lemma 7.8 (Generalized Sampling Lemma)**.** *Let $k|\ell$ and $G \subseteq \{0,1\}^{\ell n}$ be such that $|G| \geq \alpha 2^{\ell n}$.*

Consider the distribution D_G generated by first sampling an ℓ-tuple $(x_1, \ldots, x_\ell) \xleftarrow{u} G$ and a k-set $\{i_1, \ldots, i_k\} \subseteq [\ell]$ (such that $i_1, i_2, \cdots, i_k$ are distinct) and then outputting $(x_{i_1}, \ldots, x_{i_k})$.

Then,

$$\Delta(\mathcal{U}_{kn}; D_G) \leq \sqrt{2\ln(2) \cdot \frac{k \log(1/\alpha)}{\ell}}$$

where $\mathcal{U}_{kn}$ denotes the uniform distribution on kn bits and the Δ -operator corresponds to total variation distance.

The proof follows from the same techniques as that of [Raz98; IJK06] but we include it in full for convenience. Before beginning, we note the following corollary that generalizes the above to distributions with bounded min-entropy, distributions X such that $H_\infty(X) \leq 2^{-\ell n}/\alpha$ (where $H_\infty(X) \geq k \iff \forall x, \Pr[X = x] \leq 2^{-k}$). The corollary follows from the above because any distribution with min-entropy k can be written as a convex combination of sources that are uniformly distributed over sets of size 2^k.

Corollary 7.9. *Let $G \subseteq \{0,1\}^{\ell n}$ be a distribution such that forall $g \in \{0,1\}^{\ell n}$, $\Pr[G = g] \leq \frac{1}{\alpha 2^{\ell n}}$.*

Consider the distribution D_G generated by first sampling an ℓ-tuple $(x_1, \ldots, x_\ell) \leftarrow G$ and a k-set $\{i_1, \ldots, i_k\} \subseteq [\ell]$ (such that $i_1, i_2, \cdots, i_k$ are distinct) and then outputting $(x_{i_1}, \ldots, x_{i_k})$.

Then,

$$\Delta(\mathcal{U}_{kn}; D_G) \leq \sqrt{2\ln(2) \cdot \frac{k \log(1/\alpha)}{\ell}}$$

where $\mathcal{U}_{kn}$ denotes the uniform distribution on kn bits and the Δ -operator corresponds to total variation distance

The proof of Lemma 7.8 relies on the following proposition:

Proposition 7.10. *For any distribution D over $\{0,1\}^{kn}$,*

$$\Delta(D; \mathcal{U}_{kn})^2 \leq 2\ln 2(kn - H(D))$$

where $H(D)$ is the Shannon entropy of D.

Proof. We first calculate the KL divergence between D and $\mathcal{U}_{kn}$ as

$$\mathrm{KL}(D||\mathcal{U}_{kn}) = \ln(2^{kn}) + \sum_x D(x)\ln(D(x)) = \ln 2 \cdot (kn - H(X)) \,.$$

The proposition then directly follows from Pinsker's inequality $\Delta(P,Q)^2 \leq 2 \cdot \mathrm{KL}(P||Q)$. ∎

Proof of Lemma 7.8. Let $\hat{X} = (X_1, \ldots, X_\ell)$ be the joint random variables corresponding to a uniform draw from G. Let $S = \{i_1, \ldots, i_k\} \subseteq [\ell]$ be a uniformly random k-set. Consider $X = (X_{i_1}, \ldots, X_{i_k})$. By the proposition above, it suffices to show that $H(X) \geq kn - \frac{k \log(1/\alpha)}{\ell}$.

Now we consider another equivalent random process for generating X, and we will analyze that instead. We first randomly partition the variables $X_1, \ldots, X_\ell$ into ℓ/k size-k subsets $S_1, S_2, \ldots, S_{\ell/k}$, and then output a random subset S_i. This process also generates the same X by a simple symmetry argument.

Moreover, note that for every fixed partition, we know that the average entropy of X conditioned on this partition is

$$\frac{1}{\ell/k} \sum_{i=1}^{\ell/k} H(\hat{X}_{S_i}) \geq \frac{1}{\ell/k} H(\hat{X}) \,,$$

where the inequality follows from the sub-additivity of entropy. Since X is a convex combination of its conditioning on all partitions, it follows that

$$H(X) \geq \frac{1}{\ell/k} H(\hat{X}) \,.$$

Because $H(\hat{X}) = \log |G| = \ell n - \log(1/\alpha)$, we can thus deduce what we need:

$$H(X) \geq kn - \frac{k \log(1/\alpha)}{\ell}.$$

■

Next we observe a simple testing proposition that follows immediately from a (multiplicative) Chernoff bound.

Proposition 7.11. *Suppose $f(x)$ can be evaluated in time $t(n)$ where $n = |x|$.*

Given any $(x_1, \ldots, x_k) \in \{0,1\}^{nk}$ and $y_1, \ldots, y_k \in \{0,1\}^k$ such that either:

- *Case 0: for at least $2\epsilon k$ indices $i \in [k]$, $f(x_i) \neq y_i$,*
- *Case 1: for at most ϵk indices $i \in [k]$, $f(x_i) \neq y_i$*

There exists a probabilistic test, $\texttt{test}_{\epsilon,\gamma}$, that runs in time $O(t(n) \log(1/\gamma)/\epsilon)$ that correct correctly determines which case holds with probability at least $1 - \gamma$.

Finally, we are ready to state our hardness amplification result.

Theorem 7.12. *For every three constants $\alpha, \beta, \epsilon > 0$ there are constants c, c' such that the following holds. Let f be computable in time $t(n)$, and suppose that for a large enough k it holds that f^k is (ϵ, α)-non-batch-computable in time $k' \cdot t$. Then for $\ell > c \cdot k$ such that $k|\ell$, the function f^ℓ is $(\epsilon/10, 1 - \beta)$ non-batch computable in time $\ell' \cdot t$, where $\ell' = c' \cdot k'$.*

Proof. Let $\epsilon' = \epsilon/10$. Other parameters, such as γ, will be defined later.

We will prove the contrapositive of the theorem.

Let $\mathcal{A}$ violate the non-batch-computability of f^ℓ. We say $\mathcal{A}(x_1, \ldots, x_\ell; r)$ is ϵ'-close-to-correct if for at most an ϵ'-fraction $i \in [\ell]$ $\mathcal{A}(x_1, \ldots, x_\ell; r)_i \neq f(x_i)$.

Consider the reduction that $B(x_1, \ldots, x_k; r, r')$ that does the following (we differentiate the randomness r used by $\mathcal{A}$ from the other randomness used by B, r'):

1. Sample a uniformly random k-set $S = \{i_1, \ldots, i_k\} \subseteq [\ell]$. For $j = 1, \ldots, k$, set $\overline{x}_{i_j} = x_i$.
2. For $j \notin S$, sample a uniformly random $\overline{x}_j \in \{0,1\}^n$.
3. Let $\hat{y}_1 \ldots \hat{y}_\ell = \mathcal{A}(\overline{x}_1, \ldots, \overline{x}_\ell; r)$.
4. Run $\mathtt{test}_{4\epsilon',\gamma}$ to check that $y_1, \ldots, y_\ell$ are at most $8\epsilon'$ incorrect (for $\gamma < 1/50$). If so output $y_{i_1}, \ldots, y_{i_k}$, otherwise output $\perp$.

Our ultimate batch-solver, B', will run B repeatedly until an output is found (if no output found before a timeout, B' will give up).

To analyze B, we will consider its behavior on fixed sequences of internal random coins. For any fixed sequence of random coins r, we will striate input sequences into 3 different sets according to how many instances $\mathcal{A}(\cdot; r)$ solves correctly: G_r,"Good" inputs x where $\mathcal{A}(x; r)$ is $4\epsilon'$-close-to-correct; M_r, "Medium " inputs x where $\mathcal{A}(x; r)$ is not $8\epsilon'$-close-to-correct but is $4\epsilon'$-close-to-correct; B_r, "Bad" inputs x where $\mathcal{A}(x; r)$ is not $8\epsilon'$-close-to-correct.

$$G_r := \{(\overline{x}_1, \ldots, \overline{x}_\ell) \in \{0,1\}^{\ell n} : Pr_i[\mathcal{A}(\overline{x}_1, \ldots, \overline{x}_\ell)_i \neq f(\overline{x}_i)] \leq 4\epsilon'\}$$
$$M_r := \{(\overline{x}_1, \ldots, \overline{x}_\ell) \in \{0,1\}^{\ell n} : Pr_i[\mathcal{A}(\overline{x}_1, \ldots, \overline{x}_\ell)_i \neq f(\overline{x}_i)] \in (4\epsilon', 8\epsilon']\}$$
$$B_r := \{\overline{x}_1, \ldots, \overline{x}_\ell) \in \{0,1\}^{\ell n} : Pr_i[\mathcal{A}(\overline{x}_1, \ldots, \overline{x}_\ell)_i \neq f(\overline{x}_i)] > 8\epsilon'\}$$

Note that for any $x \in G_r \cup B_r$, $\mathtt{test}_{\epsilon',\gamma}$ will correctly distinguish except with probability γ. (We do not say anything about the behavior of $\mathtt{test}$ on M_r.)

$$x \in G_r \implies \Pr[\mathtt{test}_{4\epsilon',\gamma}(x, \mathcal{A}(x; r)) = 0] \leq \gamma$$
$$x \in B_r \implies \Pr[\mathtt{test}_{4\epsilon',\gamma}(x, \mathcal{A}(x; r)) = 1] \leq \gamma$$

Next, define R to be the set of "good" random coins for $\mathcal{A}$, random strings r such that G_r is large (containing a $\beta/2$ fraction of ℓ-tuples):

$$r \in R \iff \Pr_{x \xleftarrow{u} \mathcal{U}_{\ell n}}[x \in G_r] \geq \beta/2.$$

We critically note that by a standard averaging argument[37] the set of "good" random coins is not too small,

$$\Pr_{r \xleftarrow{u} \mathcal{U}}[r \in R] \geq \beta/4.$$

We want to show that if $r \in R$: (1) B will not output $\perp$ with reasonably high probability and (2) if B does not output $\perp$ it is very likely to output something that is ϵ-close-to-correct. We will also observe that if $r \notin R$ there remains relatively small probability of B making a mistake (outputting a non-$\perp$ output that is not ϵ-close-to-correct).

Before analyzing the success of our subroutine B conditioned on it outputting something (besides $\perp$), we will first analyze a related experiment. This experiment will output k-tuples with candidate solutions. We will define the experiment such that if we condition on the prefix output being a fixed tuple $x_1, \ldots, x_k$, the probability that the solution is ϵ-close-to-correct,

[37] On at least a β fraction of the inputs $\bar{x}$ we have $\Pr_{r,i}[\mathcal{A}(\bar{x}; r)_i \neq f^\ell(x)_i] < \epsilon'$. Denote this set of inputs by X. On each $\bar{x} \in X$, by Markov's inequality we have $\Pr_r[\Pr_i[\mathcal{A}(\bar{x}; r)_i \neq f^\ell(x)_i] > 4 \cdot \epsilon'] < 1/4$. So $\Pr_{\bar{x},r}[\mathcal{A}(\bar{x}; r)$ is $4 \cdot \epsilon'$-close to correct$] \geq \Pr[\bar{x} \in X] \cdot (1 - 1/4)$. On the other hand, $\Pr_{\bar{x},r}[\mathcal{A}(\bar{x}; r)$ is $4\epsilon'$-close to correct$] \leq \Pr[r \in R] \cdot \Pr[\bar{x} \in G_r] + \Pr[r \notin R] \cdot \beta/2 \leq \Pr[r \in R] + \beta/2$. Thus, $\Pr[r \in R] \geq \Pr_{\bar{x},r}[\mathcal{A}(\bar{x}; r)$ is $4\epsilon'$-close to correct$] - \beta/2 \geq 3\beta/4 - \beta/2$.

i.e. the experiment was *successful*, is identical to the probability that our subroutine was *successful*, i.e. outputs solutions that are ϵ-close-to-correct, for that particular input $x_1, \ldots, x_k$ (with fixed internal random coins r for $\mathcal{A}$).

The experiment, Exp_r, behaves as follows:

1. Sample $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_\ell) \xleftarrow{u} \{0,1\}^{\ell n}$.
2. Sample uniformly random k-set with universe $[\ell]$, $S = \{i_1, \ldots, i_k\}$.
3. Evaluate $\mathcal{A}(\overline{x}_1, \ldots, \overline{x}_\ell; r) = (y_1, \ldots, y_\ell) = \overline{y}$.
4. If $\mathtt{test}_{4\epsilon',\gamma}(\overline{x}, \overline{y}) = 1$, output $(\overline{x}_S, \overline{y}_S)$. Otherwise, output $\perp$.

Note that if we condition on the first output of Exp_r being $(x_1, \ldots, x_k) = \overline{x}_S$, then the second output of Exp_r is identically distributed to $B(x_1, \ldots, x_k; r, \mathcal{U})$ conditioned on B not outputting $\perp$.

Now let $E_{\not\perp}$ denote the event that Exp_r does not output $\perp$ and let E_c denote the event that Exp_r outputs a $9\epsilon'$-close to correct pair. We will now analyze $\Pr[E_c | E_{\not\perp}]$. Let $\overline{X}$ denote the random tuple chosen in step 1 of Exp_r.

First, consider what happens if $\overline{x}$ sampled in step 1 is not in B_r, i.e. $\overline{y}$ output by $\mathcal{A}(\cdot; r)$ is $8\epsilon'$-close-to-correct, and Exp_r outputs a non-$\perp$ value (i.e. $E_{\not\perp}$ happens). Let Z denote the random variable corresponding to the number of $j \in [k]$ such that $y_{i_j} = f(x_{i_j})$ is distributed according to a hypergeometric distribution where there is a population of size ℓ with $(1 - 8\epsilon')\ell$ successes and k draws. Consequently, $\mathbb{E}[Z] \geq (1 - 2\epsilon')k$ and concentration bounds for hypergeoemtric distributions[38] give the following, if $\epsilon' < 1/3$:

$$\forall \overline{x} \notin B_r, \Pr[\neg E_c | \overline{X} = \overline{x}, E_{\not\perp}] \geq \Pr[Z \leq (1 - 9\epsilon')k] \leq \exp(-2(\epsilon')^2 k).$$

[38] The following is taken from [Ska13]: For Z that is hypergeometic with parameters N,M,n (i.e. draw n balls without replacement from a bag of N balls containing M black ones, output the number of black balls drawn), and expectation $\mu = \mathbb{E}[Z] = \frac{nM}{N}$ and any $t > 0$, $\Pr[Z \leq \mu - tn] \leq e^{-2t^2 n}$.

Now, for $r \in R$, we can bound the success probability of Exp_r (conditioned on $E_{\not\perp}$) as follows:

$$\begin{aligned}
\Pr[E_c|E_{\not\perp}] &= \Pr[E_c, E_{\not\perp}]/\Pr[E_{\not\perp}] \\
&= \sum_{\overline{x}\in\{0,1\}^{\ell n}} \Pr[E_c|E_{\not\perp}, \overline{X}=\overline{x}]\Pr[E_{\not\perp}, \overline{X}=\overline{x}]/\Pr[E_{\not\perp}] \\
&\geq \sum_{\overline{x}\notin B_r} \Pr[E_c|E_{\not\perp}, \overline{X}=\overline{x}]\Pr[E_{\not\perp}, \overline{X}=\overline{x}]/\Pr[E_{\not\perp}] \\
&\geq \left(1-e^{-2(\epsilon')^2 k}\right) \sum_{\overline{x}\notin B_r} \frac{\Pr[E_{\not\perp}, \overline{X}=\overline{x}]}{\Pr[E_{\not\perp}]} \\
&= \left(1-e^{-2(\epsilon')^2 k}\right) \frac{\Pr[E_{\not\perp}] - \Pr[E_{\not\perp}, \overline{X}\in B_r]}{\Pr[E_{\not\perp}]} \\
&= \left(1-e^{-2(\epsilon')^2 k}\right) \left(1 - \frac{\Pr[E_{\not\perp}, \overline{X}\in B_r]}{\Pr[E_{\not\perp}]}\right) \\
&\geq \left(1-e^{-2(\epsilon')^2 k}\right) \left(1 - \frac{\Pr[E_{\not\perp}|\overline{X}\in B_r]\Pr[\overline{X}\in B_r]}{\Pr[E_{\not\perp}, \overline{X}\in G_r]}\right) \\
&\geq \left(1-e^{-2(\epsilon')^2 k}\right) \left(1 - \frac{\gamma\cdot 1}{\beta/2}\right) \\
&\geq 1 - e^{-2(\epsilon')^2 k} - \frac{2\gamma}{\beta}.
\end{aligned}$$

Let X be the random variable corresponding to the first output of Exp_r. (Recall that $\overline{X}$ is the ℓ-tuple sampled at the outset of Exp_r.) Define D_r to be the distribution of the first output of Exp_r conditioned on $E_{\not\perp}$: for $x \in \{0,1\}^{kn}$, $D_r(x) := \Pr[X = x|E_{\not\perp}]$.

We now will show that $B(\cdot; r, \mathcal{U})$ correctly solves a large fraction of its input, conditioned on it not outputting $\perp$. We will do this by comparing the behavior of the joint distribution of $(\mathcal{U}_{kn}, B(\mathcal{U}_{kn}; r, \mathcal{U}))$ to the output of Exp_r, conditioning both on not outputting $\perp$.

To invoke Corollary 7.9 on D_r, show that we can equivalently sample D_r in a manner that matches the syntax there. Observe that we can equivalently sample D_r by appropriately sampling from the right distribution over ℓ-tuples (that implicitly incorporates $\mathtt{test}$) and then directly down-sampling.

In particular, consider $\mathcal{D}_r$ the distribution on ℓ-tuples such that D_r can be sampled by (a) sampling $\overline{x} \leftarrow \mathcal{D}_r$, (b) sampling a random k-set S, and (c) outputting $\overline{x}_S$.

Recall that:

$$\begin{aligned}
D_r(x) = \Pr[X = x|E_{\not\perp}] &= \frac{\sum_{\overline{x}} \Pr[\overline{X}=\overline{x}]\cdot\Pr[\mathtt{test}(\overline{x})=1]\cdot\Pr_S[\overline{x}_S = x]}{\Pr[E_{\not\perp}]} \\
&= \sum_{\overline{x}} \underbrace{\frac{\Pr[\overline{X}=\overline{x}]\cdot\Pr[\mathtt{test}(\overline{x})=1]}{\Pr[E_{\not\perp}]}}_{\mathcal{D}(\overline{x})} \cdot \Pr_S[\overline{x}_S = x]
\end{aligned}$$

Thus, we can see that (because $\overline{X}$ is uniform over $\{0,1\}^{\ell n}$)

$$\mathcal{D}_r(\overline{x}) := \frac{\Pr[\overline{X}=\overline{x}]\cdot\Pr[\mathtt{test}(\overline{x})=1]]}{\Pr[E_{\not\perp}]} = \frac{\Pr[\overline{X}=\overline{x}]\cdot\Pr[\mathtt{test}(\overline{x})=1]]}{\sum_{\overline{x}'}\Pr[\overline{X}=\overline{x}']\Pr[\mathtt{test}(\overline{x}')=1]} = \frac{\Pr[\mathtt{test}(\overline{x})=1]}{\sum_{\overline{x}'}\Pr[\mathtt{test}(\overline{x}')=1]}.$$

Note that if $r \in R$, then for any $\overline{x}$, we can upper-bound $\mathcal{D}_r(\overline{x})$ using a lower bound on total mass on G_r. We can deduce such a bound via (a) any $\overline{x} \in G_r$ passes $\texttt{test}$ (and will not produce $\perp$) with probability at least $1-\gamma$, (b) $|G_r| \geq \beta/2 \cdot 2^{\ell n}$:

$$\mathcal{D}_r(\overline{x}) = \frac{\Pr[\texttt{test}(\overline{x}) = 1]}{\sum_{\overline{x}'} \Pr[\texttt{test}(\overline{x}) = 1]} \leq \frac{1}{(1-\gamma)|G_r|} \leq \frac{2}{(1-\gamma)\beta 2^{\ell n}}.$$

Hence, by our note on sampling D_r using $\mathcal{D}_r$, can invoke Corollary 7.9 to deduce that for $r \in R$:

$$\Delta(D_r; \mathcal{U}_{kn}) < \sqrt{2\ln(2)\frac{k \log\left(\frac{2}{(1-\gamma)\beta}\right)}{\ell}} =: \rho.$$

Recall our observation that the distribution of the second output of Exp_r conditioned on the first input being x is identical to the distribution of the output of $B(x; r, \mathcal{U})$ conditioned on it not being $\perp$. Hence, by post-processing[39] both distributions via the functionality $x \mapsto (x, B(x; r, \mathcal{U}'))|B(x; r, \mathcal{U}') \neq \perp)$, we can deduce that for $r \in R$,

$$\Delta(\mathsf{Exp}_r | E_{\not\perp}; (\mathcal{U}_{kn}, B(\mathcal{U}_{kn}; r, \mathcal{U}'))|B(\mathcal{U}_{kn}; r, \mathcal{U}') \neq \perp) \leq \rho.$$

It follows from the definition of total variation distance that the output of the statistical test that outputs 1 if the pair is $9\epsilon'$-close-to-correct differs by at most ρ between the two joint distributions above. Thus it follows,

$$\Pr_{x,r'}[B(x; r, r') \text{ is } 9\epsilon'\text{-close-to-correct} | B(x; r, r') \neq \perp] \geq 1 - e^{2(\epsilon')^2 k} - \frac{2\gamma}{\beta} - \rho.$$

Next, we would like to show that for most x, $\Pr_{r'}[B(x; r, r') \neq \perp]$ is noticeable (when $r \in R$). Let D_R denote the distribution induced by sampling D_r when $r \xleftarrow{u} R$. Similarly define the correlated random variable G_R, corresponding to G_r for the same r in D_r.

Define $H := \{x : D_R(x) \leq 2^{-(kn+1)}\}$. By our bound on the distance of D from uniform we can deduce that $|H| \leq 2^{kn+1}\rho$.

Let p_x denote the probability of $E_{\not\perp}$ (w.r.t. $r \xleftarrow{u} R$) conditioning on $\overline{X}$ containing x . Note that

$$\mathbb{E}_x[p_x] \geq \Pr[\overline{X} \in G_R] - \Pr[\texttt{test}(\overline{X}, \mathcal{A}(\overline{X}; r) = 0 | \overline{X} \in G_R] \geq \beta/2 - \gamma.$$

Moreover, $D_R(x) = \frac{p_x}{\sum_{x'} p_{x'}} = \frac{p_x}{2^n \mathbb{E}[p_{x'}]}$. So, if $x \notin H$, then $p_x \geq \beta/4 - \gamma/2$. Thus, if $r \in R$ then for $x \notin H$,

$$\Pr_{r'}[B(x; r, r') \neq \perp] \geq \beta/4 - \gamma/2.$$

Finally, $r \notin R$, we can show that nothing catastrophic is likely to happen. In particular, note that we can still bound the probability of the following events using the claims above. For any $x \in \{0,1\}^{kn}$,

$$\Pr[\ B(x; r, \mathcal{U}) \text{ does not output } \perp | \overline{X} \in B_r] \leq \gamma$$
$$\Pr[B(x; r, \mathcal{U}) \text{ is not } 9\epsilon'\text{-close-to-correct or } \perp | \overline{X} \notin B_r] \leq \exp(-2(\epsilon')^2 k) \leq \gamma;$$

where we abuse notation and take $\overline{X}$ to denote the tuple input to $\mathcal{A}$ in step 3. Thus for $r \notin R$ and any $x \in \{0,1\}^{kn}$,

$$\Pr[B(x; r, \mathcal{U}) \text{ is neither } 9\epsilon'\text{-close-to-correct nor } \perp] \leq \gamma.$$

[39] A key fact about statistical distance is that it cannot be increased by post-processing: for any randomized functionality f and any random variables A, B, $\Delta(f(A); f(B)) \leq \Delta(A; B)$.

We are now prepared to show that repeated trials will allow us to amplify the success probability of B. In particular, B' will run B q times, outputting the first non-$\perp$ output from B.

We show the probability that some trial with $r \notin R$ produces a bad (non-$9\epsilon'$-close-to-correct and non-$\perp$) output is at most $q\gamma$. For any given trial, the probability that B samples $r \in R$ and does not output $\perp$ is at least $\beta/4 \cdot (\beta/4 - \gamma/2) \geq (\beta/4)^2$.

Hence, for $x \notin H$, the probability that either B' fails to output on some round where $r \in R$ or outputs incorrectly on a round with $r \notin R$, is at most

$$(1 - \beta/4 \cdot (\beta/4 - \gamma/2))^q + q\gamma$$

Thus, for $q := O(1/\beta^2 \log(1/\alpha))$ and $\gamma := \Theta(\alpha\beta^2/\log(1/\alpha))$, B' will fail to output (something other than $\perp$) on a round where $r \in R$ with probability at most $\alpha/3$.

So, combining this, with our bound on the correctness of B for $r \in R$ conditioned on B not outputting $\perp$, as well as the bound on $|H|$ we get the following:

$$\Pr_x[B'(x) \text{ is } 9\epsilon'\text{-close-to-correct}] \geq 1 - \alpha/3 - e^{2(\epsilon')^2 k} - \frac{2\gamma}{\beta} - 3\sqrt{2\ln(2)\frac{k\log\left(\frac{2}{(1-\gamma)\beta}\right)}{\ell}}.$$

Thus, by our choice of $\gamma = \Theta(\alpha\beta^2/\log(1/\alpha))$, taking $k$to be appropriately large, ℓ to be appropriately larger $\ell \geq \Theta(k\log(2/(1-\gamma)\beta)/\alpha^2)$, we can bound

$$\Pr_x[B'(x) \text{ is } 9\epsilon'\text{-close-to-correct}] > 1 - \alpha.$$

Finally, note that B runs in time $t_B = O(t\ell' + t\log(1/\gamma)/\epsilon)$, and B' runs in time $O(q \cdot t_B)$. Thus, if α, β, ϵ are constants, then so are γ and q. As a result, the runtime of B' is simply $O(t\ell')$.

■

7.4 Putting it all together

Our main theorem for this section is an immediate corollary of Theorem 7.5 (weak non-batch-computability for tightly hard downward-self-reducible functions with efficient low-degree extension) and Theorem 7.12.

Theorem 7.13. *There is an absolute constant $c > 1$ such that the following holds. If there exists a boolean function f, a function $t : \mathbb{N} \to \mathbb{N}$, and constants $\alpha, \beta > 0$ and $\gamma' < 1$ such that*

1. *f is hard for randomized $O(t(n)^{1-\alpha})$ time (on the worst-case),*
2. *f admits a $(t(n)^{1-\alpha}, t(n), n^{o(1)})$-low-degree extension (in particular, f is computable in time t),*
3. *f is $(m, n^{\gamma'})$-downward self reducible in time $t(n)^{1-\alpha}$, where $m = m(n)$ is such that $t(m) \cdot n^{3\beta} \leq t(n)^{1-\alpha}$.*

then for some $\gamma \in (\gamma', 1)$ there is a direct product function g mapping n bits to n^γ bits such that each output bit of g is computable in time t, and g is $(.01, .99)$-non-batch-computable in time $n^{\alpha\gamma} \cdot t$.

From these observations, we have the following corollary to the theorem above:

Corollary 7.14. *If either the (randomized) strong exponential time hypothesis or the (randomized) k-orthogonal vectors conjecture for dimension $d \leq \text{polylog}(n)$ are true, then there is a function g mapping n bits to n^γ bits such that each output bit of g is computable in time $t = \tilde{O}(n^k)$, but g is $(1/100, 99/100)$-non-batchable in time $n^{\alpha\gamma} \cdot t$, for some constants $\gamma, \alpha \in (0, 1)$.*

Proof. Recall that $t(n) = \tilde{O}(n^k)$. For any $\alpha < 1/2$ and $\gamma' > 0$, by Proposition 7.4, the problem k-OV admits an $(n^{k(1-\alpha)}, \tilde{O}(n^k \cdot d), kd)$-low-degree extension, which (plugging in t) yields a $(t(n)^{1-\alpha}, t(n), n^{o(1)})$-low-degree-extension. Also, by Proposition 7.4 again with $c = \gamma'$, k-OV is $(n^{k/(k+\gamma')}, n^{\gamma'})$-downward self-reducible in time $n^{(1+\gamma')k/(\gamma'+k)}$.

We now use Theorem 7.13 with sufficiently small $\alpha, \beta > 0$. Specifically, we take $\alpha > 0$ to be sufficiently small to satisfy Item (1), relying on the k-OV conjecture (or on SETH). To satisfy Item (3), we need $t(m) \cdot n^{3\beta} \leq t(n)^{1-\alpha}$, or equivalently

$$n^{k^2/(k+\gamma')+3\beta} \leq n^{(1-\alpha)\cdot k} \ ;$$

this can be satisfied by taking $\alpha, \beta > 0$ that are sufficiently small (as functions of γ'). ■

Acknowledgements

We thank Alec Dewulf for useful comments about Section 2, and for pointing out several inaccuracies.

Marshall Ball is supported in part by a Simons Junior Faculty Fellowship. Lijie Chen is supported by a Miller Research Fellowship. Roei Tell is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2024-04490.

Part of this work was performed while the authors were visiting the Simons Institute for the Theory of Computing at UC Berkeley.

References

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

[Bat68] Kenneth E. Batcher. "Sorting Networks and Their Applications". In: *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*. Vol. 32. AFIPS Conference Proceedings. Thomson Book Company, Washington D.C., 1968, pp. 307–314.

[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.

[BGH+24] Marshall Ball, Juan A. Garay, Peter Hall, Aggelos Kiayias, and Giorgos Panagiotakos. "Towards Permissionless Consensus in the Standard Model via Fine-Grained Complexity". In: *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part II*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14921. Lecture Notes in Computer Science. Springer, 2024, pp. 113–146. DOI: 10.1007/978-3-031-68379-4_4. URL: https://doi.org/10.1007/978-3-031-68379-4_4.

[BRS+17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. "Average-case fine-grained hardness". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM, 2017, pp. 483–496. DOI: 10.1145/3055399.3055466. URL: https://doi.org/10.1145/3055399.3055466.

[BRS+18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. "Proofs of Work From Worst-Case Assumptions". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 789–819. DOI: 10.1007/978-3-319-96884-1_26. URL: https://doi.org/10.1007/978-3-319-96884-1_26.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. "On the concrete efficiency of probabilistically-checkable proofs". In: *Proc. 45th Annual ACM Symposium on Theory of Computing (STOC)*. 2013, pp. 585–594.

[BSGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. "Robust PCPs of proximity, shorter PCPs and applications to coding". In: *SIAM Journal of Computing* 36.4 (2006), pp. 889–974.

[CGI+16] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. "Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility". In: *Proc. 7th Conference on Innovations in Theoretical Computer Science (ITCS)*. 2016, pp. 261–270.

[CLO+23] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. *Polynomial-Time Pseudodeterministic Construction of Primes*. Under Submission. 2023.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. "Practical verified computation with streaming interactive proofs". In: *Proc. 3rd Conference on Innovations in Theoretical Computer Science (ITCS)*. 2012, pp. 90–112.

[CPS99] Jin-Yi Cai, Aduri Pavan, and D Sivakumar. "On the hardness of permanent". In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 1999, pp. 90–99.

[CRT22] Lijie Chen, Ron D. Rothblum, and Roei Tell. "Unstructured Hardness to Average-Case Randomness". In: *Proc. 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2022.

[CRT25] Lijie Chen, Ron D. Rothblum, and Roei Tell. "Fiat-Shamir in the Plain Model from Derandomization (Or: Do Efficient Algorithms Believe that NP = PSPACE?)" In: *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*. 2025.

[CT21a] Lijie Chen and Roei Tell. "Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise". In: *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2021, pp. 125–136.

[CT21b] Lijie Chen and Roei Tell. "Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost". In: *Proc. 53st Annual ACM Symposium on Theory of Computing (STOC)*. 2021, pp. 283–291.

[CT23a] Lijie Chen and Roei Tell. "Guest column: New ways of studying the BPP = P conjecture". In: *ACM SIGACT News* 54.2 (2023), pp. 44–69.

[CT23b] Lijie Chen and Roei Tell. "When Arthur has Neither Random Coins nor Time to Spare: Superfast Derandomization of Proof Systems". In: *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*. 2023.

[CTW23] Lijie Chen, Roei Tell, and R. Ryan Williams. *Derandomization vs Refutation: A Unified Framework for Characterizing Derandomization*. Under Submission. 2023.

[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, pp. 1057–1068.

[DPT24] Dean Doron, Edward Pyne, and Roei Tell. "Opening up the distinguisher: a hardness to randomness approach for BPL=L that uses properties of BPL". In: *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*. 2024, pp. 2039–2049.

[DT23] Dean Doron and Roei Tell. "Derandomization with Minimal Memory Footprint". In: *Electronic Colloquium on Computational Complexity: ECCC* 30 (2023), p. 036.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for muggles". In: *Journal of the ACM* 62.4 (2015), 27:1–27:64.

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

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

[Gol11] 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.

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

[IJK06] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. "Approximately List-Decoding Direct Product Codes and Uniform Hardness Amplification". In: *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*. IEEE Computer Society, 2006, pp. 187–196. DOI: 10.1109/FOCS.2006.13. URL: https://doi.org/10.1109/FOCS.2006.13.

[IJK07] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. "Chernoff-Type Direct Product Theorems". In: *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*. Ed. by Alfred Menezes. Vol. 4622. Lecture Notes in Computer Science. Springer, 2007, pp. 500–516. DOI: 10.1007/978-3-540-74143-5_28. URL: https://doi.org/10.1007/978-3-540-74143-5_28.

[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.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which problems have strongly exponential complexity?" In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530.

[IW97] 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)*. 1997, pp. 220–229.

[IW98] Russell Impagliazzo and Avi Wigderson. "Randomness vs. Time: De-Randomization under a Uniform Assumption". In: *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1998, pp. 734–743.

[KM02] Adam R. Klivans and Dieter van Melkebeek. "Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses". In: *SIAM J. Comput.* 31.5 (2002), pp. 1501–1526.

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

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

[LP22b] Yanyi Liu and Rafael Pass. "Leakage-Resilient Hardness v.s. Randomness". In: *Electronic Colloquium on Computational Complexity: ECCC* TR22-113 (2022).

[LPT24] Jiatu Li, Edward Pyne, and Roei Tell. "Distinguishing, Predicting, and Certifying: On the Long Reach of Partial Notions of Pseudorandomness". In: *Proc. 65th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2024.

[MMS23] Dieter van Melkebeek and Nicollas Mocelin Sdroievski. "Instance-wise hardness versus randomness tradeoffs for Arthur-Merlin protocols". In: *Proc. 38th Annual IEEE Conference on Computational Complexity (CCC)*. Vol. 264. LIPIcs. Leibniz Int. Proc. Inform. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2023, Art. No. 17, 36. DOI: 10.4230/lipics.ccc.2023.17. URL: https://doi.org/10.4230/lipics.ccc.2023.17.

[MS23] Dieter van Melkebeek and Nicollas Sdroievski. "Leakage resilience, targeted pseudorandom generators, and mild derandomization of Arthur-Merlin protocols". In: *Proc. 43rd Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 284. LIPIcs. Leibniz Int. Proc. Inform. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2023, Art. No. 29, 22.

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

[PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. "Certified hardness vs. randomness for log-space". In: *Proc. 64th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. [2023] ©2023, pp. 989–1007.

[Raz98] Ran Raz. "A Parallel Repetition Theorem". In: *SIAM J. Comput.* 27.3 (1998), pp. 763–803. DOI: 10.1137/S0097539795280895. URL: https://doi.org/10.1137/S0097539795280895.

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

[She92] Alexander Shen. "IP = PSPACE: Simplified Proof". In: *J. ACM* 39.4 (1992), pp. 878–880. DOI: 10.1145/146585.146613. URL: https://doi.org/10.1145/146585.146613.

[Ska13] Matthew Skala. *Hypergeometric tail inequalities: ending the insanity*. 2013. arXiv: 1311.5939 [math.PR]. URL: https://arxiv.org/abs/1311.5939.

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

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

[SV22] Ronen Shaltiel and Emanuele Viola. "On hardness assumptions needed for "extreme high-end" PRGs and fast derandomization". In: *Proc. 13th Conference on Innovations in Theoretical Computer Science (ITCS)*. 2022, Art. No. 116, 17.

[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.

[Val79] Leslie G Valiant. "The complexity of computing the permanent". In: *Theoretical computer science* 8.2 (1979), pp. 189–201.

[WB86] Lloyd R Welch and Elwyn R Berlekamp. *Error correction for algebraic block codes*. US Patent 4,633,470. 1986.

[Wil15] Virginia V. Williams. "Hardness of easy problems: basing hardness on popular conjectures such as the Strong Exponential Time Hypothesis". In: *Proc. 10th International Symposium on Parameterized and Exact Computation*. Vol. 43. 2015, pp. 17–29.

[Wil18] Virginia Vassilevska Williams. "On some fine-grained questions in algorithms and complexity". In: *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*. World Scientific. 2018, pp. 3447–3487.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il