Electronic Colloquium on Computational Complexity, Report No. 345 (2026)

Targeted Samplers

Heda Chen* Dean Doron† M. H. Ebtehaj‡ Roei Tell§

24 Nov 2026

Abstract

We introduce a new pseudorandom object whose purpose is to reduce the error of a probabilistic algorithm as efficiently as possible. This object is called a targeted sampler, and it is a version of a standard general-purpose sampler in which the event that we want to sample is explicitly given to the targeted sampler (in the form of a given input to a uniform probabilistic algorithm).

A key potential in studying these objects is that they can have a smaller sample size than general-purpose samplers. This feature is useful for superfast derandomization, since it allows reducing an algorithm's error while incurring little time overhead. Our main contributions are:

1. **Quantified derandomization.** We conditionally show that reducing a time-T algorithm's error to $\delta(T) = 2^{-\Omega(T)}$ suffices to derandomize the algorithm entirely while incurring essentially no time overhead. Our quantified derandomization of algorithms with error δ relies on hardness for probabilistic non-deterministic algorithms, and improves on prior work.
2. **Constructions.** We show three conditional constructions of targeted samplers and hitters that reduce the error of an algorithm to δ as above, while incurring essentially no time overhead. (In comparison, general-purpose samplers incur a quadratic time overhead for such error-reduction.) Each of these constructions is based on a different hardness assumption.
3. **Codes with a doubly-local decoder.** Towards deducing the foregoing hardness assumptions from standard assumptions, we introduce a notion of doubly-local decoding of an error-correcting code: this is a decoder whose running time is polylogarithmic in both the block length and in the (inverse of) the agreement parameter, using a non-deterministic oracle. We show two new constructions of such decoders, based on high-dimensional expanders.

Our contributions can be combined to obtain conditional derandomization of probabilistic algorithms that incurs essentially no time overhead. More generally, our work puts forward a highly modular approach for studying such derandomization: we identify and separate the key challenging parts in the process, define new pseudorandom objects for studying each of these parts (targeted sampler, doubly-local decoder), and present constructions of these objects.

*The University of Toronto, Department of Computer Science. Email: hedachen03@gmail.com
†Ben Gurion University, Faculty of Computer and Information Science. Email: deand@bgu.ac.il
‡The University of Waterloo, Institute for Quantum Computing. Email: mhebtehaj@uwaterloo.ca
§The University of Toronto, Department of Computer Science. Email: roei@cs.toronto.edu

ISSN 1433-8092

Contents

1 Introduction **1**
- 1.1 The notion of targeted samplers . . . 2
- 1.2 Setting the stage: Better quantified derandomization . . . 5
- 1.3 Constructions of targeted samplers . . . 5
- 1.4 Codes with a doubly-local decoder . . . 8
- 1.5 Discussion: Implications to overhead-free derandomization . . . 10

2 Technical Overview **12**
- 2.1 Better quantified derandomization . . . 12
- 2.2 Targeted hitter from hard circuits with low-degree gates . . . 14
- 2.3 Targeted hitter from hardness of meta-complexity . . . 16
- 2.4 Codes with a doubly-local decoder . . . 17

3 Preliminaries **19**

4 Better Quantified Derandomization **19**
- 4.1 Unconditional overhead-free error-reduction . . . 20
- 4.2 Two-sided error vs one-sided error in quantified derandomization . . . 25
- 4.3 Quantified derandomization from uniform hardness assumptions . . . 27

5 Targeted Samplers Based on Non-batch-computable Functions **44**

6 Targeted Hitters Based on Circuits with Low-degree Gates **47**
- 6.1 A reconstructive targeted generator . . . 48
- 6.2 The targeted hitter and error-reduction . . . 58

7 Targeted Hitters Based on Hardness of Probabilistic Levin-Kolmogorov Complexity **59**
- 7.1 A reconstructive targeted generator . . . 60
- 7.2 The targeted hitter and error-reduction . . . 63

8 Codes with a Doubly-Local Decoder **65**
- 8.1 The code $\mathcal{C}_{\mathrm{HDX}}$ . . . 66
- 8.2 An inner code $\mathcal{C}_{\mathrm{in}}$ . . . 72
- 8.3 The concatenated code . . . 75

9 Open Problems **79**

A On Overhead-Free Derandomization **86**

B Another Construction of a Condenser from Sub-constant Rate **88**

C Deferred Proofs of Technical Claims **95**

1 Introduction

Given a probabilistic algorithm A that errs with probability at most $1/3$ on any given input, how can we obtain another algorithm A' solving the same problem with much smaller error (say, error 2^{-n} on n-bit inputs)? For simplicity, and without loss of generality, we focus on algorithms solving decision problems, rather than search problems (see [Gol11b; Tel24; Gol24] for search-to-decision reductions).

The naive approach repeats A multiple times and takes the majority vote among the outcomes. A more randomness-efficient approach, which is standard by now, uses an object called a sampler to obtain a sample S of inputs, and takes the majority vote among the outcomes of A on S (for example, S can be obtained by a random walk on an expander; see [Gol11a] for a survey).[1] It is well-known that samplers are equivalent to seeded randomness extractors (see, e.g., [Vad12, Corollary 6.24]).

The observation underlying this paper is that the standard sampler-based approach is an overkill for this task. Specifically, a sampler has the property that for *every* Boolean test f of the sampler's outputs S, with high probability over the randomness of the sampler, the average of f on S is close to the true average of f. However, when trying to reduce the error of a *specific* algorithm A, we are not concerned with arbitrary information-theoretic tests f: We are only concerned with a specific test $f = A$, and moreover, we have a *description of the specific test at our disposal*.[2]

In other words, when reducing the error of a probabilistic algorithm, instead of using a sampler, which approximately samples all information-theoretic tests, we only need an object that approximately samples a specific and explicitly given test.

Our contributions, at a high level. We introduce a new pseudorandom object called targeted sampler, which we argue is the appropriate object for error-reduction of probabilistic algorithms. Intuitively, a targeted sampler gets a description of a test f, and with high probability over its coins, produces a small sample S such that the average of f (specifically) on S is close to the true average of f.

1. Delineating the potential in the study of this object, in Section 1.1 we identify a main possible advantage, which is having targeted samplers with a small sample size. This advantage is particularly useful when studying "superfast derandomization", which incurs as little overhead as possible.

2. Setting up the stage: quantified derandomization. We point out a particularly interesting setting for studying targeted samplers, by conditionally showing that reducing a time-T algorithm's error to $\delta(T) = 2^{-\Omega(T)}$ would suffice to derandomize the algorithm entirely while incurring essentially no time overhead.

[1] For brevity, when mentioning samplers in this introduction we refer to *averaging* samplers. There are other known types of samplers, which in fact yield better parameters than those that are known for averaging samplers; see [Gol11a] (as well as the recent [XZ25]) for further details.

[2] To be more accurate, we know the description of the machine A, and we have an explicit input x (on which we want to compute the function that A computes, but with very low error), and the relevant test is $f(\cdot) = A(x, \cdot)$, where the second input to A represents its choice of random coins.

Specifically, in Section 1.2 we show overhead-free derandomization of algorithms whose error is at most δ, relying on hardness assumptions for uniform non-deterministic algorithms. This improves on prior work by handling a smaller δ than before and by relying on hardness only for uniform procedures.

3. Three constructions. In Section 1.3 we present three conditional constructions of targeted samplers with error $\delta(T) = 2^{-\Omega(T)}$. Each of the constructions relies on a different hardness assumption, where these assumptions are non-standard but weaker than corresponding assumptions required for full derandomization.

   Technically, two of these constructions leverage new reconstructive targeted generators that we design for this purpose and that may be of independent interest, whereas one construction relies on a known targeted generator.

4. Codes with a doubly-local decoder. To bridge the gap between the non-standard assumptions used in our constructions and more standard hardness assumptions, in Section 1.4 we introduce a new object called an error-correcting code with a doubly-local decoder. We present two codes with doubly-local decoders, based on high-dimensional expanders.

Finally, in Section 1.5 we explain the implications of the new notions and constructions to superfast derandomization. In particular, we explain how our work introduces a highly modular approach to this task, and re-opens the question of whether it is possible to obtain overhead-free derandomization that is correct in the worst-case.

Remark 1.1. *Our work includes a relatively large number of distinct technical contributions, which may be of independent interest. We will mention these very briefly throughout the introduction, and elaborate in the technical overview in Section 2.*

1.1 The notion of targeted samplers

We first formally define targeted samplers, and delineate the possibilities and limitations in studying this notion.

Recall that a standard sampler gets randomness r and outputs a sample-set such that for every function f, with high probability $1 - \delta$ over r, the expected value of f on the sample-set is approximately the same as the expected value of f on a uniform string, up to a small error ϵ. A targeted sampler is defined similarly, except that it also gets input $x \in \{0,1\}^n$, and refers to a computationally bounded class of functions, which in our case is uniform algorithms running in time $T(n)$ on x.[3]

Definition 1.2 (targeted sampler)**.** *A* targeted sampler for T-time algorithms with accuracy $\epsilon(\cdot)$ and error $\delta(\cdot)$ *is an algorithm*

$$\mathsf{tarSamp}\colon \{0,1\}^n \times \{0,1\}^{T'} \to (\{0,1\}^T)^D$$

[3]Indeed, the sampler's output length $T = T(n)$ matches the algorithm's running time $T(n)$ on $x \in \{0,1\}^n$, since the output of the targeted sampler is intended to replace the algorithm's randomness.

such that for every T-time algorithm A and all but finitely many $x \in \{0,1\}^n$, with probability at least $1-\delta(T')$ over the choice of $r \in \{0,1\}^{T'}$ it holds that

$$\left|\mathbb{E}_{y\in\{0,1\}^T}[A(x,y)] - \mathbb{E}_{s\in[D]}[A(x,\mathsf{tarSamp}(x,r)_s)]\right| \leq \epsilon(T)\ . \tag{1.1}$$

We call T' the randomness complexity and D the sample complexity.

For simplicity, throughout the introduction we fix $\epsilon = .01$. The reason that we denote the randomness length by T' is the expected application of error-reduction for algorithms. Indeed, if we start with a probabilistic algorithm A, we can define a new algorithm $A'(x,r) = \mathsf{MAJ}\left\{A(x,\mathsf{tarSamp}(x,r)_s)\right\}_{s\in[D]}$ that solves the same problem as A using T' random coins and having error at most $\delta(T')$ at every input x.

How can targeted samplers beat general samplers? We have constructions of excellent general (i.e., non-targeted) samplers, which use $T' = T + O(\log(1/\delta))$ random coins, have sample complexity $D = O(\log(1/\delta))$, and run in linear time. In fact, up to the constants hidden inside the O-notations, these constructions are optimal (see [Gol11a] for a survey). Hence, potential advantages of targeted samplers may come from *having smaller randomness complexity T' and/or smaller sample size D*, beating the information-theoretic lower bounds for general samplers.

Are targeted samplers easier to build than targeted PRGs? Targeted pseudorandom generators (PRGs), as defined by Goldreich [Gol11b], are the special case of targeted samplers that have randomness zero, i.e. $T' = 0$. Hence, targeted samplers are a *considerably weaker pseudorandom object*. Moreover, since efficient targeted PRGs with a tiny sample size $D = O_\epsilon(1)$ exist if and only if $pr\mathcal{BPP} = pr\mathcal{P}$ (see [Gol11b]), we expect targeted samplers with $D = O_\epsilon(1)$ to exist too, with any value of T'.[4]

One may be tempted to suggest a path of gradually improving T': Start with known general samplers that use $T' = T + O(\log(1/\delta))$, and gradually decrease T' until we reach $T' = 0$, obtaining targeted PRGs and $pr\mathcal{BPP} = pr\mathcal{P}$. However, this path is unfortunately an overkill: even a mild improvement over general samplers already yields unconditional non-trivial general derandomization, rather than only error-reduction (which is the intended purpose of targeted samplers). That is:

Observation 1.3. *A targeted sampler for T-time algorithms with $T' < T - \log(D)$ and $\delta \leq 1/4$ yields a targeted PRG for T-time algorithms with an output-set of size less than 2^T.*

Proof. The targeted PRG outputs $G(x) = \{\mathsf{tarSamp}(x,r)_s\}_{r\in\{0,1\}^{T'},s\in[D]}$. ■

[4] To be more accurate, if the targeted PRG/sampler is required to fool a specific T-time algorithm (e.g., for a complete problem) then it can have a sample-set of size $D = O_\epsilon(1)$. If we want it to fool all T-time algorithms, then a union-bound over the first $f(n) = \omega(1)$ machines allows for $D = O_\epsilon(\log(f(n)))$. Since both values of D are tiny, we ignore the difference throughout the exposition.

The promising avenue: small sample size and superfast derandomization. We thus focus on targeted samplers that use more than T random coins (e.g., $T' = O(T)$) but whose sample size is $D \ll \log(1/\delta)$, beating general samplers.

Such targeted samplers are particularly useful for studying superfast derandomization, as introduced in [DMO+20]. This is the question of derandomizing algorithms while incurring as little resource overhead as possible (e.g., no time and space overhead), which attracted significant recent attention (see, e.g., [CT21b; CT21a; SV22; CT23b; DT23; DPT24; LPT24; BCT25; DMO+26]). The reason targeted samplers with $D \ll \log(1/\delta)$ can be useful for this purpose is that enumerating over the sample set (for error-reduction) incurs a multiplicative time overhead of D, but if $D \ll \log(1/\delta)$ this overhead is smaller than when using general samplers.

In more detail, a main application for studying targeted samplers with small D is the following. Recall that the classical non-uniform derandomization of Adleman [Adl78] simulates T-time algorithms by circuits of size $O(n \cdot T)$. Recent works conditionally matched this by uniform algorithms [DMO+20; CT21b], and it is a major open question to determine if the multiplicative time overhead of n (which is costly for, say, linear-time algorithms) is necessary. This is where targeted samplers come in:

Observation 1.4. *If we construct targeted samplers with $\delta(T') = 2^{-\Omega(T')}$ and $D = O(1)$ and linear running time, then $pr\mathcal{BPTIME}[T'] \subseteq \mathcal{SIZE}[O(T')]$.*

Proof sketch. Use naive error-reduction with constantly many repetitions to further decrease the error to be less than $2^{-T'} < 2^{-n}$, at which point we can hard-wire a single random string that is good for all n-bit inputs.[5] ■

That is, constructing such targeted samplers would allow us to avoid the overhead in non-uniform derandomization. As we will show in the next section, we can also conditionally match the foregoing derandomization by *uniform algorithms*. (For comparison, with general samplers, getting error $\delta(T') = 2^{-\Omega(T')}$ requires $D = \Omega(\log(1/\delta) = \Omega(T')$, which incurs a quadratic time overhead for error-reduction.) Thus,

> **A main application of targeted samplers:** The question of whether derandomization with no time overhead is possible reduces to the question of whether targeted samplers can have smaller sample complexity than general samplers; that is, whether "looking inside the distinguisher" (i.e., receiving x) helps reduce the sample size below $\log(1/\delta)$.

Remark 1.5. *A* targeted hitter *is defined by a straightforward modification of Definition 1.2. Specifically, we replace Eq. (1.1) with the following condition: If $\mathbb{E}_{y \in \{0,1\}^T}[A(x,y)] \geq \epsilon(T)$,*

[5] Indeed, this argument works even with the larger error $\delta = 2^{-\Omega(n)}$. In this paper we measure the error as a function of the number of random coins T' (as is done throughout the literature studying samplers, error-reduction, and quantified derandomization).

then there is $s \in [D]$ such that $A(x, \mathsf{tarSamp}(x,r)_s) = 1$. We can use a targeted hitter to reduce the error of probabilistic algorithms that have only one-sided error.

1.2 Setting the stage: Better quantified derandomization

In this section we further motivate the study of targeted samplers with $\delta(T') = 2^{-\Omega(T')}$ and $D \ll \log(1/\delta)$ by showing, under hardness assumptions, that such samplers would imply overhead-free derandomization.

Specifically, we consider the question of quantified derandomization, which is the derandomization of algorithms that only have a tiny error of δ, introduced by Goldreich and Wigderson [GW13] (for a survey see [Tel22]). Previous works showed overhead-free quantified derandomization only from smaller $\delta(T) = 2^{T^{.99}-T}$ and assuming lower bounds for non-uniform non-deterministic circuits (see [CT21b]).

We improve prior works on both fronts: we conditionally show derandomization from a larger $\delta(T) = 2^{-\Omega(T)}$ as we wanted, and the hardness assumption is only for *uniform* probabilistic non-deterministic algorithms (rather than for non-deterministic circuits). This addresses open problems posed in [Tel22, Open Problems 1 & 2].

Theorem 1.6 (overhead-free quantified derandomization from $\delta(T) = 2^{-\Omega(T)}$ under uniform hardness assumptions; informal, see Theorem 4.30)**.** *Assume that for every $T(n) = \mathrm{poly}(n)$ there is $f : \{0,1\}^n \to \{0,1\}^n$ computable by uniform circuits of size $\bar{T} = T(n)^{1+O(\epsilon)}$ and depth $\mathrm{polylog}(T)$ such that f is hard for probabilistic non-deterministic algorithms[6] running in time $\bar{T}^{1+O(\epsilon)}$ and space $\bar{T}^{1-\Theta(\epsilon^4)}$ on all but finitely many inputs. Then, for every $T(n) = \mathrm{poly}(n)$,*

$$\mathsf{BPTIME}_{2^{(1-\epsilon)T}}[T] \subseteq \mathsf{DTIME}[T^{1+O(\epsilon)}] \,,$$

where $\mathsf{BPTIME}_Q[T]$ refers to problems solvable with error probability $\delta(T) = Q/2^T$.

There are several new technical contributions going into Theorem 1.6, which may be of independent interest. We show an unconditional overhead-free error-reduction from $\delta(T) = 2^{-\Omega(T)}$ to $\delta(T) \approx 2^{T^{.99}-T}$; we also show a reduction of quantified derandomization of algorithms with two-sided error to quantified derandomization of algorithms with one-sided error, which incurs almost no time overhead (unlike the reduction for general algorithms); and we construct a new targeted hitting-set generator for quantified derandomization based on hardness as stated in the theorem (this generator is, in turn, based on a new bootstrapping system a-la [CT21a] that has a stronger property than in prior works, which we refer to as a "locality" property).

1.3 Constructions of targeted samplers

We show three constructions of targeted samplers/hitters with $D \ll \log(1/\delta)$ (i.e., not necessarily $D = O(1)$ but still very small), based on three different assumptions.

[6] Formally, we assume hardness against a subclass of $\mathsf{MA} \cap \mathsf{AM}$; see Definitions 4.19 and 4.20.

1.3.1 Hardness on all x_1 and on $1-\delta$ of x_2

The first two constructions are based on assumptions of the form: for all x_1, for $1-\delta$ of x_2, a probabilistic algorithm A fails to compute a hard function f on input $x = (x_1, x_2)$. In the pair (x_1, x_2) the first input x_1 (on which we require "for all" hardness) is short $|x_1| = n$ and the second input x_2 (on which we require "$1-\delta$" hardness) is long $|x_2| = T' = O(T)$; hence, x_2 dominates the input length $O(T)$.

This form of pair-hardness seems ad-hoc, and in particular the parameter value $\delta(T') = 2^{-\Omega(T')}$ in the condition $1-\delta$ on x_2 seems unjustified. We provide justification in the next section, where we make progress on bridging the gap between this assumption and more standard ones. For now, note that this form of hardness is more relaxed than hardness assumptions required for full derandomization, which require hardness on all inputs x to the hard function f.

Construction I: Non-batch-computability. We start by stating a result that can be obtained easily from known technical tools, specifically from the targeted PRG of Chen and Tell [CT21a]. As we will explain next, this result will not be satisfactory for our later purposes, and therefore we state it here briefly and informally.

Theorem 1.7 (targeted sampler from non-batch-computability; informal, see Corollary 5.4)**.** *Assume that OWFs exist, and that there is $f : \{0,1\}^n \times \{0,1\}^{O(T)} \to \{0,1\}^{k=n^\epsilon}$ whose individual output bits are computable in time $T' = T \cdot n^\epsilon$, but for all x_1 and for $1-\delta$ of x_2, the k-bit output cannot be approximately printed in time $k^{.01} \cdot T'$. Then, there is a targeted sampler for T-time algorithms that has error δ, sample complexity $n^{O(\epsilon)}$, and randomness complexity $O(T)$, whose sample-set is computable in time $T \cdot n^{O(\epsilon)}$.*

The assumption in Theorem 1.7 is more relaxed than the assumption made in [CT21a], which yielded a targeted PRG, because hardness is required only on a $(1-\delta)$-fraction of $x_2 \in \{0,1\}^{O(T)}$.[7] However, the non-batch-computability hardness required in Theorem 1.7 is not standard, and in particular, it is not clear how to obtain such hardness from more standard assumptions via hardness amplification (which is the direction we pursue in Section 1.4). This motivates our search for additional constructions.

Construction II (main new construction): Small circuits with "big" gates. Our main construction relies on hard functions that are structured, in the sense that the function is computable by a certain type of uniform circuit. The upside is that the hardness we require is standard, and it will fit into the hardness amplification paradigm.

Specifically, we consider functions computable by small circuits (e.g., of size $n^\epsilon \times n^\epsilon$) whose gates are, essentially, uniform arithmetic $\mathbb{F}_2$-circuits of size $\approx T$. This model can be considered as a generalization of arithmetic circuits, where we wire together several arithmetic circuits, each of them low-degree by itself.

[7] To be more accurate, in [CT21a] the assumption referred to all polynomial-time samplable distributions over x_1 rather than to every x_1, but Theorem 1.7 also extends to support such an assumption; see Remark 1.10. The main new point is that hardness is required only on $1-\delta$ of x_2.

Definition 1.8 (uniform circuits with low-degree gates; see Definition 6.1)**.** *We say that a circuit family $\{C_N\}$ has $(\Delta, \bar{T})$-low-degree gates if every gate in C_N can be computed by an arithmetic circuit over $\mathbb{F}_2$ of degree Δ and size $\bar{T}$ that can be printed in time $\tilde{O}(\bar{T})$. We also require that there is a uniform family of low-depth formulas that, given (g, j), outputs the index of the jth child of gate g.*

The hardness assumption in the following result refers to uniform circuits of dimension $k \times k$, so the total number of gates is k^2. Intuitively, each gate is a low-degree arithmetic circuit of size $\bar{T}$. It is trivial to compute such circuits in time $\tilde{O}(k^2 \cdot \bar{T})$, and the assumption is that such circuits cannot be computed in time $k^{.01} \cdot \bar{T}$.

Theorem 1.9 (targeted hitter from hardness of uniform circuits with low-degree gates; informal, see Corollary 6.10)**.** *Assume that OWFs exist, and that there is a family of circuits $C = \left\{C_{n+O(T)} \colon \{0,1\}^{n+O(T)} \to \{0,1\}^{k=T^{O(\epsilon)}}\right\}$ of dimensions $k \times k$ with $(T^{\epsilon^2}, \bar{T} = T^{1+O(\epsilon)})$-low-degree gates such that the following holds. For every randomized algorithm running in time $k^{1+\epsilon} \cdot \bar{T}$, for all but finitely many $x_1 \in \{0,1\}^n$, for $1 - \delta$ of x_2, the algorithm fails to compute $C_{n+O(T)}(x_1, x_2)$. Then, there exists a targeted hitter for T-time algorithms that has error δ, randomness complexity $O(T)$, sample complexity $T^{O(\epsilon)}$, and runs in time $T^{1+O(\epsilon)}$.*

Theorem 1.9 is based on a construction of a new targeted hitting-set generator. In fact, this is the first new construction of a targeted generator that allows superfast derandomization since the work of Chen and Tell [CT21a], and the first such construction that is based on new technical machinery. (The previous construction was technically based solely on [NW94].)

Remark 1.10. *In Theorems 1.7 and 1.9 the hardness assumptions can be replaced by hardness assumptions asserting "for all polynomial-time samplable distributions on x_1, with all but negligible probably over x_1, with probability $1 - \delta$ over a uniform x_2, the hardness holds". (In fact, both theorems translate hardness with respect to every distribution on x_1 to targeted samplers that work correctly over that distribution.) We mention this to align our work with prior works, which assumed hardness over all samplable distributions and deduced derandomization over all samplable distributions. We further explain this point in Section 1.5.*

1.3.2 Construction III: Hardness of estimating probabilistic Levin-Kolmogorov complexity

We next present another construction, which relies on hardness of estimating conditional probabilistic Levin-Kolmogorov complexity, denoted $\mathsf{cpKt}_{1-\delta}$.

For context, recall that the Levin-Kolmogorov complexity of w, denoted $\mathsf{Kt}(w)$, is the size of the smallest program that outputs w in bounded time.[8] The conditional variant $\mathsf{cKt}(w|x)$ is defined analogously, except that the small program is given x as input. Assuming the existence of targeted PRGs fooling linear-time algorithms (in particular, assuming $pr\mathcal{BPP} = pr\mathcal{P}$), every linear-time algorithm that gets (x, w)

[8] Specifically, $\mathsf{Kt}(w)$ is the minimal k such that there is a program of size s outputting w in 2^{k-s} steps.

fails to distinguish between the case that $\mathsf{cKt}(w|x) \leq O(\log|w|)$ and the case that $\mathsf{cKt}(w|x) \geq |w| - O(\log|w|)$, for all but finitely many x. This is because outputs w of the targeted PRG on x have small $\mathsf{cKt}(w|x)$ whereas a random string has high $\mathsf{cKt}(w|x)$, and the targeted PRG fools linear-time algorithms (see, e.g., [LP22]).

We will construct a targeted hitter relying on an assumption that is weaker than the one needed for targeted PRGs, and that refers to *probabilistic* cKt. Recall that $\mathsf{pKt}_{1-\delta}(w)$, introduced by Goldberg *et al.* [GKL+22], is essentially the Levin-Kolmogorov complexity of w when randomness is allowed, and the success probability needs to be at least $1-\delta$. Specifically, and adding the condition x, they defined $\mathsf{cpKt}_{1-\delta}(w|x)$ as the smallest k such that with probability at least $1-\delta$ over a random string r there is a program Π of size $s \leq k$ such that $\Pi(x,r) = w$ in at most 2^{k-s} steps.

The construction below relies on hardness of distinguishing between $\mathsf{cpKt}_{1-\delta}(w|x) \leq (1+\epsilon)\cdot\log(|w|)$ and $\mathsf{cpKt}_{1-\delta}(w|x) \geq |w| - |w|^{\epsilon}$. This appears to be a harder problem than that of distinguishing between low and high cKt (i.e., than the problem whose hardness is needed for targeted PRGs), because the sets of "yes" instances is larger.[9] The parameter δ, which is the main one making the problem harder, determines the error of the targeted hitter. We will take δ to be tiny, yielding a tiny error.[10]

Theorem 1.11 (targeted hitter based on hardness of approximating cpKt; informal, see Corollary 7.5)**.** *For $T(n) \geq n$ and $\delta(|w|) = 2^{-\Omega(|w|^{1+\epsilon})}$, assume that every linear-time algorithm that is given $(x,w) \in \{0,1\}^n \times \{0,1\}^{T^{1+\epsilon}}$ fails to distinguish between the case that $\mathsf{cpKt}_{1-\delta}(w|x) \leq (1+\epsilon)\cdot\log(|w|)$ and the case that $\mathsf{cpKt}_{1-\delta}(w|x) \geq |w| - O(|w|^{\epsilon})$, for all but finitely many x's.[11] Then, there is a targeted hitter for T-time algorithms that has error δ, randomness complexity $T^{1+O(\epsilon)}$, sample complexity $T^{O(\epsilon)}$, and runs in time $T^{1+O(\epsilon)}$.*

Interestingly, Theorem 1.11, does *not* rely on one-way functions (i.e., it bypasses the "hybrid argument" barrier; see Section 1.5). Similarly to Remark 1.10, a relaxed assumption referring to hardness over a distribution over inputs x yields a corresponding targeted hitter that works over the same distribution.

1.4 Codes with a doubly-local decoder

As mentioned in Section 1.3.1, the form of hardness assumption used there – "for all x_1, with probability $1-\delta$ over x_2" – was unjustified, and in particular the use of the specific value $\delta(T) = 2^{-\Omega(T)}$ was unjustified. We now provide justification for such δ, and introduce a new coding-theoretic object for studying the assumption.

Consider a function $f(x_1,x_2)$ that is hard for all x_1 with probability $1/100$ (rather than $1-\delta$) over x_2. We consider this to be a more standard form of assumption,

[9] The two problems do not seem strictly comparable, because in cpKt the set of "no" instances is also larger. However, in both problems the set of "no" instances simply consists of the vast majority of strings, and it is natural to suspect that the hardness mainly comes from recognizing "yes" instances.

[10] Specifically, in the statement below we take $\delta = 2^{-\Omega(|w|^{1+\epsilon})}$. In the "yes" case, the minimal non-trivial value of δ is $2^{-|w|^{1+\epsilon}}$, and the assumption refers to $\delta = 2^{-c\cdot|w|^{1+\epsilon}}$ for a constant $c > 0$. Also, the reason the "yes" case refers to $(1+\epsilon)\cdot\log(|w|$ rather than $O(\log|w|)$ is that we want a near-linear-time hitter.

[11] That is, there are only finitely many x such that the algorithm succeeds for all $w \in \{0,1\}^{T(|x|)}$.

or at the very least more relaxed. We would like to use standard tools of hardness amplification, to amplify the hardness to be for $1-\delta$ of x_2's. Specifically, for every fixed x_1, consider the function $f_{x_1}(\cdot)$, and let us try and apply standard hardness amplification tools to it. We apply an error-correcting code to the truth-table of f_{x_1}, and in particular use a code with local encoding (without loss of generality, a direct-product code), to preserve the upper bound on computing f_{x_1}.

We need the code to have polynomial rate (so that the length of the second input of the encoded function is $O(|x_2|) = O(T)$), which is standard. And to complete the standard hardness amplification argument, we need the code to be locally list-decodable from agreement $\delta(T) = 2^{-\Omega(T)}$. Specifically, since we are concerned with hardness for algorithms running in time $\approx T$, we want the list-decoder to run in time $\mathrm{poly}(T)$, and ideally in time $O(T)$.

Codes with a doubly-local decoder. It is the last requirement that creates a problem: every known code we are aware of has a decoder that runs in time at least $1/\delta$, which in our setting is exponential in T. This seems inherent: Intuitively, a decoder from agreement δ distinguishes between the all-zero word and a word of weight δ, and in our setting of an exponentially small δ the satisfiability problem can be reduced to such a task; thus, we do not expect a decoder running in time $\ll 1/\delta$ to exist.[12]

Nevertheless, we show that *non-determinism is the only obstacle*. Specifically, we construct codes whose decoder runs in time $\mathrm{polylog}(N, 1/\delta)$ while using oracle queries to $\mathcal{NP}$. We call this a doubly-local decoder, since its running time is polylogarithmic both in the block length N and in the (inverse of) the agreement parameter δ.

Definition 1.12 (codes with a doubly-local decoder)**.** *We say that an error-correcting code* $\mathsf{Enc}\colon [q_1]^K \to [q_2]^N$ *has a* doubly-local list decoder from agreement δ *if there is an oracle machine* $\mathsf{Dec}(i, \alpha)$ *that is a standard local list decoder from agreement* δ,[13] *and runs in time* $\mathrm{polylog}(N, 1/\delta)$ *while making queries to its oracle.*

We use capital-letter notations such as K and N to remind the reader that in our setting the message and codeword are truth-tables of size $K = 2^{O(T)}$ and (when the rate is polynomial) $N = 2^{O(T)}$. The definition extends to *approximate* doubly-local list-decoders in the straightforward way (i.e., as usual, requiring that only $1-\mu$ of the coordinates of the codeword will be correctly decoded for a small $\mu > 0$).

In retrospect, a construction of Drucker [Dru13] meets this definition, achieving an approximate doubly-local list-decoder with an $\mathcal{NP}$ oracle from $\delta(N) \approx 2^{-\sqrt{\log(N)/\log(K)}}$ (e.g., with a large quasipolynomial rate $N = 2^{(\log K)^C}$, the agreement can be as small as $\delta(N) = 2^{-(\log N)^{.49}}$). We are interested in a smaller $\delta(N) = N^{-\Omega(1)} = 2^{-\Omega(\log N)}$, which matches the sought hardness amplification parameter δ from Section 1.3.1.

[12] For direct-product codes, Dikstein *et al.* [DHP+26, Theorem 7.1] proved that decoders indeed need at least $1/\delta$ queries. We are not aware of such a result for the general case. (The obstacle to a straightforward proof is the fact that the decoder is given an index, which helps distinguish between any two codewords.)

[13] That is, for every $w \in [q_2]^N$ and every codeword c in the ball of radius $1-\delta$ around w there is α such that $\mathsf{Dec}(i, \alpha) = c_i$ for all i. As usual, we focus on codes with list size $L = \mathrm{poly}(1/\delta)$ and $|\alpha| = O(\log L)$.

We show two codes that significantly improve on this prior work. The first code achieves the δ that we want, and thus provides partial justification for the hardness assumptions in Section 1.3.1; however, this code has too large an alphabet, whose log-size is $\text{poly}(1/\delta)$. The second code reduces the alphabet to log-size $O(\log(1/\delta))$, but supports only a larger $\delta(N) \approx 2^{-\sqrt{\log N}}$.

Theorem 1.13 (a code with a doubly-local decoder from an exponentially-small agreement; informal, see Theorem 8.3)**.** *For any $\delta(N) \geq N^{-\Omega(1)}$ there is a code $\Sigma_0^K \to \Sigma^N$ with rate $N = \text{poly}(1/\delta) \cdot \tilde{O}(N)$ and alphabet log-size $\log(|\Sigma|) = \text{poly}(1/\delta, \log N) \cdot \log(|\Sigma_0|)$ that has a .01-approximate doubly-local list decoder from agreement δ with list-size $O(1/\delta)$, running time $\text{polylog}(N, 1/\delta)$, and access to an $\mathcal{NP}^w$ oracle where w is the corrupt codeword. Moreover, treating Σ as a tuple of Σ_0-symbols, each of the latter symbols can be computed from the message in time $\text{polylog}(N \cdot |\Sigma_0|/\delta)$.*

Theorem 1.14 (a code with a doubly-local decoder from agreement $\delta \approx 2^{-\sqrt{\log N}}$; informal, see Theorem 8.6)**.** *For any $\delta(K) \geq K^{-\Omega(1)}$ there is a code $\Sigma_0^K \to \Sigma^N$ with rate satisfying $\log(N) = \log(K) + \log(1/\delta)^2 \cdot O(\text{loglog}(|\Sigma_0| \cdot K))$ and alphabet log-size $\log(|\Sigma|) = O(\log(N/\delta))$ that has a .01-approximate doubly-local list decoder from agreement δ with list-size $\tilde{O}(1/\delta)$, running time $\text{polylog}(N, 1/\delta)$, and access to an Σ_2^w oracle. Moreover, each codeword symbol can be computed from the message in time $\text{polylog}(N \cdot |\Sigma_0|/\delta)$.*

The codes underlying Theorems 1.13 and 1.14 are the high-dimensional expanders code of Dikstein *et al.* [DHP+26], and a concatenation of this code with the standard direct-product code and a uniquely decodable code (again following an idea in [DHP+26]). The main new parts in both results is designing a doubly-local decoder.

Theorems 1.13 and 1.14 do not fully bridge the gap between standard hardness assumptions and the required assumption in Theorem 1.9, because the codes' parameters are not optimal; see Section 9 for an open problem.[14] However, Theorem 1.14 does support an extreme form of hardness amplification, which assumes hardness on .01 of inputs for polynomial-time algorithms with a Σ_2-oracle, and deduces hardness for standard polynomial-time algorithms on $\approx 1 - 2^{-\sqrt{n}}$ fraction of n-bit inputs.

1.5 Discussion: Implications to overhead-free derandomization

The study of superfast derandomization is challenging, and in particular it has been fraught with ad-hoc assumptions, and its technical toolkit is still not developed enough (as recently explained in [BCT25]). Our results add three contributions to this study.

Richer technical toolkit, and new results. Our work contributes new technical tools, in particular the new targeted generator for quantified derandomization underlying Theorem 1.6 and the new targeted generator underlying the hitter construction in Theorem 1.9; both new generators incur essentially no time overhead.

[14] In addition, to fully plug this into our setting of $f(x_1, x_2)$, we would also need a method to deduce the list-index for the decoder in the reduction. We view this issue as secondary to the more natural question of codes with a doubly-local decoder.

In addition, our work deduces new conditional superfast derandomization results, which are obtained by combining any targeted sampler or hitter from Section 1.3 with the quantified derandomization algorithm from Theorem 1.6. (In fact, the latter results deduce derandomization with almost no overhead that is correct in the worst-case; we will explain this in a moment.)

A highly modular approach. Our work isolates the core challenging parts in superfast derandomization, introduces new objects for studying each part (targeted samplers and doubly-local decoders), and presents constructions of such objects.

In particular, prior works focused on constructing targeted generators.[15] Our work partitions this task into two separate tasks: reducing the error to $\delta(T) = 2^{-\Omega(T)}$, and quantified derandomization of algorithms with error δ. Since the latter task seems more tractable (by Theorem 1.6 and since overhead-free non-uniform derandomization is possible), *our work isolates the core difficulty in superfast derandomization as lying in the task of error-reduction*. The notion we introduce – targeted samplers – precisely addresses this core difficulty. We further identify hardness ampification as a challenge in constructing targeted samplers, and introduce doubly-local decoders to study it.

Re-opening the question of overhead-free derandomization. Prior works conditionally deduced worst-case derandomization that has a multiplicative time overhead of $T \mapsto n \cdot T$ (see [DMO+20; CT21b]), and also conditionally deduced derandomization that is correct on average (say, over all polynomial-time distributions) and incurs very small time overhead, say of $T \mapsto n^{\epsilon} \cdot T$ for an arbitrarily small $\epsilon > 0$ (see [CT21a; BCT25]). In terms of terminology, the former was called "superfast derandomization" whereas the latter was called "free lunch derandomization".

Part of the rationale underlying prior works was the impression that worst-case derandomization with no overhead might be impossible, backed by partial evidence. Let us call such derandomization "overhead free derandomization". We suggest to treat the foregoing impression with suspicion. Indeed, *the results obtained in our work imply overhead-free derandomization*, and (as mentioned in Section 1.1) the question of whether overhead-free derandomization is possible reduces to the question of whether targeted samplers can have a smaller sample than general samplers (which seems like an appealing possibility).[16] In Appendix A we re-examine the known evidence for impossibility of overhead-free derandomization, with the intention to cast doubt on it.

[15] One challenge within this task is bypassing the "hybrid argument barrier" (i.e., transformation of distinguishers to next-bit predictors). This challenge can be bypassed by assuming one-way functions, which seem to be an overkill but are standard. For further details see the recent [DMO+26].

[16] There is another possibility: it could be that targeted samplers with a sample of size $D = O_{\epsilon}(1)$ do exist, but are only computable in time $O(n \cdot T)$ rather than $O(T)$. This is not ludicrous: Goldreich's conditional targeted PRG [Gol11c] runs in time $O(n \cdot T)$ and has a sample of size $O_{\epsilon}(1)$.

2 Technical Overview

There is a large number of distinct technical contributions in the paper, but fortunately most of them can be read independently of each other. In particular, each of the subsections in the current section, which correspond to Sections 1.2, 1.3.1, 1.3.2 and 1.4 respectively, is self-contained.

2.1 Better quantified derandomization

Our goal is to prove Theorem 1.6, which derandomizes any probabilistic T-time algorithm A that has error $2^{-\Omega(T)}$ at every input, relying on uniform hardness assumptions and while increasing the runtime only to $T^{1+O(\epsilon)}$. The construction has three steps, and we need to ensure neither step adds significant time overhead.

Overhead-free error-reduction. We would like to reduce A's error from $\delta(T) = 2^{-\Omega(T)}$ to $\delta'(T') = 2^{(T')^{.99}-T'}$. If we use general-purpose samplers, the sample size will be $\log(1/\delta') = \Omega(T)$ and the error-reduction will incur a quadratic time overhead (since the new algorithm takes a majority of A over the sample).

It is well-known that general-purpose samplers are equivalent to randomness extractors [Zuc97]. We observe that a weaker version of an extractor – specifically, a randomness condenser – can reduce the error from an initial small value (of $2^{-\Omega(T)}$) to δ (see Theorem 4.15), where the key point is that such a condenser does not need to incur a quadratic slowdown. In standard condenser terminology, we allow a linear entropy gap $g = \Omega(T)$ for a T-bit output, in which case there are non-explicit condensers with seed length $\log(\frac{T-k}{g}) = O(1)$ (cf., extractors require $\log(T-k)$; see [AT19]).

We show that there are explicit condensers that can be computed in *near-linear* time and support error-reduction from $2^{-\Omega(T)}$ to $\delta'(T')$ with time overhead $T^{O(\epsilon)}$ (i.e., seed length $\epsilon \cdot \log(T)$; see Theorem 4.14). In fact, we show two constructions: one combines a somewhere-condenser of Zuckerman [Zuc06] with the Dvir-Wigderson merger [DW08; DKS+13], and another construction uses an alternative somewhere-condenser based on the one by Raz [Raz05] (see Section 4 and Appendix B).

Overhead-free reduction of two-sided error to one-sided error. The construction we will show later on only derandomizes probabilistic algorithms with one-sided error. It is well-known that derandomization of algorithms with two-sided error reduces to derandomization of algorithms with one-sided error (see, e.g., [Sip83; Lau83; ACR98; BF99; GVW11; GZ11]), but known reductions incur a large polynomial time overhead. Even worse, for non-black-box derandomization (as ours will be) the time overhead increases further to be compositional $T \mapsto \text{poly}(T(\text{poly}(T)))$.

Our main observation is that this overhead can be completely avoided when derandomizing algorithms with tiny error $\delta'(T') = 2^{(T')^{.99}-T'}$. Specifically, we observe that in the reduction of Goldreich and Zuckerman [GZ11], the time overhead comes almost

entirely from a preliminary step of reducing general derandomization to quantified derandomization (i.e., of algorithms with tiny error), and after that the remaining part is essentially overhead free, and reduces the resulting quantified derandomization problem to (a combination of) two quantified derandomization problems for algorithms with one-sided error. In particular, if we start from a quantified derandomization problem, the last step lets us reduce it to quantified derandomization of algorithms with one-sided error while incurring almost no time overhead. See Lemma 4.16.

An overhead-free targeted hitting-set generator. Finally, we want to solve the quantified derandomization problem for algorithms with one-sided error and error probability $\delta'(T')$, while incurring almost no overhead.

Towards this purpose, we construct a new targeted hitting-set generator whose correctness is based on the hardness assumption in Theorem 1.6. Let us start with a naive idea that will fail and will be fixed later. Recall that the hard function is computable by a uniform circuit C of size $\bar{T} = T^{1+\epsilon}$ and depth $d = \text{polylog}(T)$. Following known ideas [Sip88; MV05; DMO+20; CT21b], on input x the generator computes the Boolean matrix of gate-values in $C(x)$, splits each row into T^ϵ strings of length T', and outputs the $d \cdot T^\epsilon$ strings. Intuitively, if all of these strings are "easy" then reconstructing the computation $C(x)$ is also "easy".

In more detail, assume that all of these strings fall in the tiny set S of $2^{(T')^{.99}}$ random strings that cause our algorithm to mistakenly reject.[17] A probabilistic non-deterministic algorithm can then compute $C(x)$ as follows. As a first step, it randomly chooses a hash function h shrinking strings of length T' to be of length $(T')^{.999}$, and with high probability there are no collisions in S. (Note that this choice is made before any non-deterministic choice, and all we will need is that there are no collisions in S.)

Then, for each row $r^{(i)}$ in $C(x)$, from bottom to top, the algorithm computes a compressed version of the row, in the form of T^ϵ hash values $h(r_1^{(i)}), ..., h(r_{T^\epsilon}^{(i)})$, corresponding to the T^ϵ substrings $r_1^{(i)}, ..., r_{T^\epsilon}^{(i)}$ in row i. The compressed version can be stored using only $T^\epsilon \cdot (T')^{.999} < \bar{T}$ space, and it allows us to later non-deterministically de-hash every $r_j^{(i)}$: we can ask the prover to send a string $\tilde{r}$ that is supposedly $r_j^{(i)}$, and we check that $\tilde{r} \in S$ and that $h(\tilde{r})$ matches the stored hash (which guarantees correctness when there are no collisions in S). Going iteratively from bottom to top, we compute each substring in $r^{(i+1)}$ using the fact that $r^{(i+1)}$ is a simple function of $r^{(i)}$ (and using the compressed version of $r^{(i)}$ that we stored), and store its hash, obtaining a compressed version of $r^{(i+1)}$ (and then throwing away the compression of $r^{(i)}$ and proceeding). In the end we would have computed the output row of $C(x)$ using space less than $\bar{T}$ (because we only store two compressed versions at each point).

This idea fails, because it is inefficient. The reason is that computing all of $r^{(i+1)}$ in the iterative step requires a total of $\bar{T}$ queries to $r^{(i)}$, and each non-deterministic de-hashing operation costs time $\approx T'$. Overall the time complexity is quadratic $\approx \bar{T}^{2+O(\epsilon)}$,

[17] For simplicity we stick with the simplified error probability $2^{(T')^{.99}-T'}$. In actuality this error probability will be $2^{(T')^{1-c\cdot\epsilon}-T'}$ for some $c > 1$, and we choose $\bar{T} = T^{1+C\cdot\epsilon}$ with a sufficiently large $C > c$.

and we would have to assume hardness against quadratic-time algorithms.

The problem would go away if $r^{(i+1)}$ would depend on $r^{(i)}$ in a nice way, for example if every substring in $r^{(i+1)}$ would depend only on the substring of $r^{(i)}$ directly below it. This is because we can then de-hash the substring of $r^{(i)}$ below *once*, and use it to compute the corresponding substring of $r^{(i+1)}$, paying time $\approx T'$ to compute T' entries of $r^{(i+1)}$. It turns out that something like this can be achieved!

We define a notion of a bootstrapping system with block locality, which is an encoding of a gate-values matrix $C(x)$ such that every substring in each row $r^{(i+1)}$ depends only on a small number (five, to be precise) of substrings in the row $r^{(i)}$ below it. We show how to encode $C(x)$ into a bootstrapping system with block locality without significantly increasing its size or depth. The technical idea is to sort the entries in the rows using a highly structured sorting network (specifically, following Ball, Chen, and Tell [BCT25], we use Batcher's [Bat68] bitonic sort). See Theorem 4.23 for details, and Section 4.3.3 for the resulting generator and derandomization.

2.2 Targeted hitter from hard circuits with low-degree gates

Recall that we want to construct targeted samplers and hitters with error δ from hardness of some $f(x_1, x_2)$ on all x_1 and for $1 - \delta$ of x_2. (In Section 2.4 we describe the hardness amplification justifying such an assumption.) We do so using reconstructive targeted pseudorandom generators and hitting-set generators. Specifically, the latter generators, based on a hard function f, transform hardness of f on any fixed input $x = (x_1, x_2)$ into pseudorandomness (for some predetermined uniform algorithm) on the same input $x = (x_1, x_2)$. In particular, if f is hard on all x_1 and for $1 - \delta$ of x_2, we obtain a generator whose pseudorandomness is guaranteed on all x_1 and for $1 - \delta$ of x_2. And now, if we think of $x_1 = x$ as input and of $x_2 = r$ as randomness, this generator G is itself, by definition, a targeted sampler/hitter $G(x, r)$ with error δ.[18]

The main tool in proving Theorem 1.9 is a new targeted hitting-set generator (HSG) that is based on hard uniform circuits with low-degree gates (as in Definition 1.8), and that incurs essentially no time overhead; specifically, the generator is computable in near-linear time and has sub-linearly many output strings (equivalently, a sub-logarithmic seed length). This generator, presented in Section 6.1, is the first generator that incurs no time overhead after the initial construction of Chen and Tell [CT21a].

Main ideas. Our starting point is the construction in [CT21a]. Their generator relied on a hard function $f_0(x)$ that outputs a k-bit string, where each output bit is computable in time $\bar{T}$ but the entire string cannot be approximately batch-computed in time $k^{.01} \cdot \bar{T}$. The generator itself applied the classical NW [NW94] to the k-bit string (considered as a truth-table), and if this does not fool a uniform T-time algorithm at x, then the string can be batch-computed in time $k^{.01} \cdot \bar{T}$, contradicting the hardness of f. (This relies on a parameter choice of $\bar{T} = k \cdot T = T^{1+O(\epsilon)}$, for a small $k = T^{O(\epsilon)}$.)

[18] In fact, it even meets a stronger requirement, in which the distinguisher is also allowed to be aware of r (rather than only of x).

Let us think of this k-bit string as a bottom layer in a circuit. A bottleneck is that the "reconstruction" algorithm (which computes the k-bit string if the generator is not pseudorandom) only *approximately* computes $f_0(x)$, and in particular a constant fraction of the k bits that it outputs are erroneous. Indeed, if we would have been able to error-correct the string $f_0(x)$ – that is, if the outputs of f_0 were codewords in an error-correcting code – then we could easily correct the erroneous bits.

The motivating observation is that if we think of f_0 as computing k values of a low-degree arithmetic circuit C over $\mathbb{F}_2$ on various parts of x, then $f_0(x)$ can be made error-correctible.[19] Specifically, we first encode the input x by a standard low-degree extension $\hat{x}$ over a suitably large extension field $\mathbb{F}$ of $\mathbb{F}_2$. Then, assuming that deciding the connectivity between gates and inputs is simple enough and can be arithmetized to a low-degree polynomial $\hat{\Phi}_1$ over $\mathbb{F}$, we can define $P_1 \colon \mathbb{F}^m \to \mathbb{F}$ as $P_1(v) = C(\hat{x}(\hat{\Phi}_1(v,1)), ..., \hat{x}(\hat{\Phi}_1(v,m)))$, which is low-degree hence error-correctible.

The most encouraging part is that this opens the door to recursion. Specifically, it allows for a construction based on a hard function that has more layers above the initial one, each with arithmetic circuit gates. Starting with a $k \times k$ circuit, the generator applies [NW94] to each polynomial P_i, and if the generator fails, then the reconstruction can iteratively compute each row, going from bottom to top (indeed, the key point is that it computes each row entirely, rather than approximating it). Using k rows, and recalling that the reconstruction works in time approximately $k^{.01} \cdot \bar{T}$ per row, we deduce that if the generator fails then the reconstruction computes the final row in time $k^{1.01} \cdot \bar{T}$ (which is far better than the obvious upper-bound of $\tilde{O}(k^2 \cdot \bar{T})$).

Technical subtleties and caveats. The arithmetization of the $k \times k$ circuit with low-degree gates is actually not straightforward. The problem is that when recursing we define the polynomial P_{i+1} in terms of P_i; that is, $P_{i+1}(v) = C(P_i(\hat{\Phi}_i(v,1)), ..., P_i(\hat{\Phi}_i(v,T)))$. This causes a degree blow-up we cannot tolerate. The trick to get around this, following Ball, Chen, and Tell [BCT25], is to add "linearization polynomials" a-la Shen [She92] between each P_i and P_{i+1}, to control for the degree blow-up. That is, we will have a sequence of polynomials that looks like $P_1, P_{1,1}, ..., P_{1,m}, P_2, P_{2,1}, ..., P_{2,m}, ..., P_k$ where m is the number of variables and each $P_{i,j}$ reduces the degree of the jth variable. The generator applies [NW94] to each P_i and $P_{i,j}$, and the reconstruction works from bottom to top, while reconstructing not only the "primary" polynomials P_i but also intermediary polynomials $P_{i,j}$ (and indeed, adding the intermediary polynomials maintains the downward self-reducibility relationship in the sequence).

A naive implementation has $m = k$ variables and adds too many polynomials. This can be reduced to $m \leq \log(T)$ using standard arithmetization techniques (i.e., we define the P_i's as polynomials not $\mathbb{F}^k \to \mathbb{F}$ but $\mathbb{F}^m \to \mathbb{F}$ for $m \approx \log(T)$, and the degree will reduce not to linear but to be low enough, i.e. $T^{\Omega(1)}$; this requires suitable arithmetizations of C and of the Φ_i's). The reconstruction then runs in time $k^{1.01} \cdot m \cdot \bar{T}$, still far lower than the upper-bound $\tilde{O}(k^2 \cdot \bar{T})$. Further details appear in Section 6.

[19] We assume for simplicity that the circuit C is identical at every gate, and only the wiring to the inputs changes from gate to gate. This assumption loses less generality than one might expect (see Remark 6.2).

2.3 Targeted hitter from hardness of meta-complexity

To prove Theorem 1.11, our goal is to construct a targeted hitter based on hardness of estimating $\mathsf{cpKt}_{1-\delta}$. As a starting point, recall the construction of a targeted HSG by Liu and Pass [LP22] based on hard of estimating cKt (i.e., the non-probabilistic version). Their generator gets input x and outputs all strings w with low $\mathsf{cKt}(w|x)$, by enumerating over machines with small description and running each machine on input x for a suitable number of steps. Loosely speaking, the reason their generator works is that a distinguisher that rejects all of its outputs and accepts a random string rejects, by definition, every w with low $\mathsf{cKt}(w|x)$, and accepts most strings.[20]

We will imitate this argument in the probabilistic setting, and use a different analysis to carry it through. Specifically, our targeted hitter on input x and randomness r outputs all strings w with low $\mathsf{cKt}(w|x,r)$. The reason we expect this to work is that if $D(x,\cdot)$ rejects, with probability $\delta' > \delta$ over r, all the outputs of the hitter on (x,r), then $D(x,\cdot)$ must reject all strings w with low $\mathsf{cpKt}_{1-\delta}(w|x)$ (because every such string appears among the outputs of the hitter for at least a $(1-\delta)$-fraction of the r's).

Now, how do we prove that such $D(x,\cdot)$ rejects *only* strings w with low $\mathsf{cpKt}(w|x)$? Fixing any w that D rejects, we will show that it has $\mathsf{cpKt}(w|x) \ll |w|$. The main technical tool will be a coding theorem by Lu, Oliveira, and Zimand [LOZ22], showing that if a string w can be sampled using x in polynomial time with probability $\epsilon > 0$, then $\mathsf{cpKt}_{1-\delta}(w|x) \lesssim \log(1/\epsilon) + O(\log(n \cdot |w|))$ (see Lemma 7.3). Since $D(x,\cdot)$ accepts most strings but rejects w, we can use rejection sampling to sample w with non-trivial probability $\epsilon = c \cdot 2^{-|w|}$ for $c > 1$, and deduce that $\mathsf{cpKt}_{1-\delta}(w|x)$ is less than $|w|$.

The complication. This idea will work, but not as-is. The problem is that the rejection sampling above beats the naive sampler only by a constant multiplicative factor, and this yields savings in $\mathsf{cpKt}_{1-\delta}$ that are too small (i.e., $\log(1/\epsilon) = |w| - O(1)$, and the bound becomes $\mathsf{cpKt}_{1-\delta}(w|x) \leq |w| - O(1) + O(\log(n \cdot |w|))$, which is trivial).

We fix this by having the hitter find all strings w with low $\mathsf{cKt}(w|x,r)$ that are of length $k \cdot T$ (rather than of length T as before); it splits each such w into k substrings $w_1, ..., w_k$ of length T, and outputs the set $\{w_i\}_{i \in [k]}$. Given $D(x,\cdot)$ that breaks the hitter, we consider an algorithm for $\mathsf{cpKt}_{1-\delta}$ that, given w, splits it into substrings $w_1, ..., w_k$, and decides that the $\mathsf{cpKt}_{1-\delta}$ complexity is low if and only if $D(x,\cdot)$ rejects every w_i.

The first part of the argument extends to this setting in a straightforward way.[21] The advantage is that in the second part of the argument, we can sample w by performing rejection sampling on each w_i *separately*, and hence the non-trivial sampling probability becomes $\epsilon = (c \cdot 2^{-T})^k = c^k \cdot 2^{-kT} = 2^{\Theta(k)-|w|}$. In particular, using $k = |w|^\epsilon$ we obtain non-trivial savings $\mathsf{cpKt}_{1-\delta}(w|x) \leq |w| - \Theta(|w|^\epsilon) + O(\log(n \cdot |w|))$ (which

[20] To complete the argument they need to show that the distinguisher accepts *every* string w with high $\mathsf{cKt}(w|x)$. They do this by modifying the generator construction (applying a classical PRG to each output of the basic generator), and we ignore this part for simplicity of presentation.

[21] Specifically, consider $D(x,\cdot)$ that rejects, with probability $\delta' > \delta$ over r, all the outputs of the hitter on (x,r). Then $D(x,\cdot)$ must reject all substrings $w_1, ..., w_k$ of strings w with low $\mathsf{cpKt}_{1-\delta}(w|x)$, because $w_1, ..., w_k$ are outputs of the hitter for at least a $(1-\delta)$-fraction of the r's.

completes the correctness argument), and the output-set of the hitter is still of small size $k = T^\epsilon$. For further details, see the proof of Proposition 7.4.

2.4 Codes with a doubly-local decoder

Our goal is to construct codes $C\colon \{0,1\}^K \to \Sigma^N$ whose decoder locally list-decodes from agreement $\delta > 0$ in time $\mathrm{polylog}(N, 1/\delta)$, with the aid of an $\mathcal{NP}$ oracle.[22]

The basic idea. At a high level, we are looking for opportunities to "shortcut" expensive decoding steps, using the $\mathcal{NP}$ oracle. One such opportunity is that many decoders involve steps of randomly choosing a set of $O(1/\delta)$ candidate objects, and testing each candidate to find a suitable one (indeed, a good one can be verified once found). A natural idea is to use an $\mathcal{NP}$ oracle to perform a binary search over the set, in time $\mathrm{polylog}(1/\delta)$, instead of the exhaustive search.

An illustrative example is the decoder of Impagliazzo *et al.* [IJK+10] for the s-wise direct product code, where $N = \binom{K}{s}$. Recall that the list-decoding index is $A \subseteq [K]$ of size $s/2$ and a local function $v\colon A \to \{0,1\}$. Given $x \in [K]$, the decoder repeatedly chooses at random a set $B \supset A \cup \{x\}$ of size s, and once it finds B such that the corrupt codeword's value $\tilde{f}(B)$ agrees with v on A, it outputs $\tilde{f}(B)|_x$. The analysis shows that (for a suitable A and correct v) a random B satisfies $\tilde{f}(B)|_A = v(A)$ with probability at least $\approx \delta$ and in this case $\tilde{f}(B)|_x$ is correct with high probability, so $O(1/\delta)$ repetitions suffice to guarantee their decoder's success.

Loosely speaking, our decoder can receive x, fix some list of $O(1/\delta)$ candidate sets B, and use the $\mathcal{NP}$ oracle to find the *lexicographically-first* B in the list such that $\tilde{f}(B)|_A = v(A)$. This requires a concise representation of the list (since we cannot store $O(1/\delta)$ sets), which can be obtained by choosing the list using a pairwise-independent generator (i.e., the generator's seed is a concise representation).

The only caveat is that the analysis does not guarantee that *any* consistent B in the list will be correct on x, but only that enough B's are consistent and most consistent B's are correct on most x's. A-priori, the lexicographically-first B containing x might be wrong on x (and more generally, a list of $O(1/\delta)$ candidates might contain a few misleading candidates). We will solve this problem differently for each of our codes.

The HDX code of [DHP+26]. The code of Theorem 1.13 is the main construction of Dikstein *et al.* [DHP+26]. This is a direct-product code where the set-system comes from a high-dimensional expander.

Their decoder relies on what they called a unit-congestion routing scheme. This is an algorithm that gets an initial set s_0 and $f_0\colon s_0 \to \{0,1\}$ (analogous to A and v in the example above) and $x \in [K]$, and finds a sequence of sets $s_1, \ldots, s_t$ where $x \in s_t$ and the following holds: for every s_i there is a list $\mathcal{L}_i$ of $O(1/\delta)$ functions on s_i such

[22] Throughout this part of the overview we keep the notation δ for the agreement parameter, to be consistent with the rest of the paper. In the coding literature the agreement parameter is denoted by $\epsilon > 0$, and we follow the convention of using $\epsilon > 0$ in Section 8.

that for any sequence $\{f_i \in \mathcal{L}_i\}_{i\in[t]}$ that approximately agree on the intersections (i.e., each f_i, f_{i+1} approximately agree on $s_i \cap s_{i+1}$), the final f_t is correct on most $x \in s_t$. Crucially, the sequence is of length $t = \text{polylog}(N)$ independently of δ.

Starting from f_0 given as advice, for $i = 1, ..., t$ our decoder iteratively uses the $\mathcal{NP}$ oracle to find the lex-first $f_i \in \mathcal{L}_i$ that approximately agrees with f_{i-1} on $s_{i-1} \cap s_i$. This takes $\text{polylog}(N)$ steps, and the binary search at each step takes time $\text{polylog}(1/\delta)$. (And indeed, each list $\mathcal{L}_i$ can be generated using a bounded-independence generator, which gives us a concise representation.)

At first this seems to run into the same problem as above, because s_t depends on x and f_t depends on s_t, and it seems like for every x we might choose an f_t that is wrong on x. To avoid this we extend a subtle trick from [IJK+10; DHP+26]. Specifically, our $\mathcal{NP}$ oracle will not know x, but rather only the path $P = (s_0, .., s_t)$, and hence for every P there is a unique $f_t \in \mathcal{L}_t$ used for decoding every $x \in s_t$. Now, to show that for most x's the value of $f_t(x)$ is correct, we observe that a random choice of $x \in [K]$ is equivalent to a random choice of P and $x \in s_t$ (this exploits the regularity of the set-system, which yields regularity of the routing scheme).[23] Now, as mentioned above, for any consistent sequence $\{f_i \in \mathcal{L}_i\}$, the final f_t is correct on most $x \in s_t$. Hence, with high probability over P and $x \in s_t$ (equivalently, with high probability over a random $x \in [K]$) the decoder is correct.

Concatenating the HDX code with the complete complex. The code of Theorem 1.14 is another construction from [DHP+26], which concatenates the HDX code with the standard direct-product code. (The inner code is actually the composition of direct-product with a uniquely decodable code, but we ignore the latter for simplicity.)

We will design a doubly-local decoder with an $\mathcal{NP}$ oracle for the direct-product code. Then, the decoder for the outer code issues oracle queries to an $\mathcal{NP}$ oracle that needs access to the simulated corrupt codeword, and each query to the latter will be resolved by the decoder for the direct-product code, which also needs an $\mathcal{NP}$ oracle. The composition gives a doubly-local decoder using a Σ_2 oracle.

To decode direct-product, we could try the same approach as in the illustrative example (i.e., take the lex-first consistent B), but that runs into the dependency issue mentioned there. The trick to get around this is to take the lex-first *index of an attempt* at choosing B that is consistent; that is, for a sequence of $O(1/\delta)$ attempts $B_1, ..., B_{O(1/\delta)} \supseteq A \cup \{x\}$, we take the consistent B_i with minimal i. This works because it precisely mimics the algorithm of [IJK+10], and one can show that choosing the $O(1/\delta)$ sets from an pairwise-independence generator (which allows us to find i using a binary search and the $\mathcal{NP}$ oracle) maintains the property that for 99% of coordinates x, with high probability, the first consistent B_i is correct on x.[24] See further details in Section 8.

[23] To be more accurate about terminology, given x the algorithm chooses s_t, and this graph is regular and allows switching the order of choice in the analysis. The routing scheme itself is the algorithm that (randomly) constructs P between s_0 and s_t.

[24] Intuitively, this property is established by showing that in any sequence of indices $1, ..., i$ we are more likely to encounter a consistent B_j that is correct on x than a consistent B_j that is incorrect on x. This

Remark 2.1. *We will actually need not only list-decoding but also list-recovery, i.e. we need doubly-local decoders for list-recoverable codes. (This is since we are using the Guruswami–Indyk [GI03] reduction, and composing approximately list-decodable codes. In fact, list-recovery is the reason we have the $\mathcal{L}_i$'s in the first code above.) The ideas above can be extended to the list-recovery setting, by using the $\mathcal{NP}$ oracle to find a canonical good index in the relevant list.*

3 Preliminaries

The machine model used throughout this paper is the Random Access Machines (RAM) model. We recall basic definitions and results regarding pseudorandomness and pseudorandom generators.

Definition 3.1 (distinguisher)**.** *A function $D\colon \{0,1\}^n \to \{0,1\}$ is an ϵ-*distinguisher *for a distribution $\mathbf{w}$ over $\{0,1\}^n$ if*

$$\left| \Pr_{r\in\{0,1\}^n}[D(r)=1] - \Pr_{w\sim\mathbf{w}}[D(w)=1] \right| \geq \epsilon\,.$$

We say that D is an ϵ-distinguisher for a multiset $S \subseteq \{0,1\}^n$ if D is an ϵ-distinguisher for the uniform distribution over S.

Definition 3.2 (pseudorandom generator)**.** *A function $G : \{0,1\}^{\ell(n)} \to \{0,1\}^n$ is an ϵ-*PRG secure against a class *$\mathcal{C} \subseteq \{\{0,1\}^n \to \{0,1\}\}$ if no $C \in \mathcal{C}$ is an ϵ-distinguisher for $G(\mathcal{U}_\ell)$.*

We will rely on the fact that if non-uniformly secure OWFs exist, then there is a PRG computable in near-linear time secure against polynomial-sized circuits.

Theorem 3.3 ([GL89; HIL+99]; See, e.g., Theorem 3.4 in [CT21a])**.** *Assume existence of OWFs secure against polynomial-size circuits. Then, for any constant $\mu > 0$, there exists an $(n^{-\omega(1)})$-PRG $G : \{0,1\}^{n^\mu} \to \{0,1\}^n$ that is computable in time $n^{1+\mu}$ and secure against polynomial-sized circuits.*

4 Better Quantified Derandomization

In Section 4.1 we show an unconditional overhead-free error-reduction from error $2^{-\Omega(T)}$ to error $2^{T^{.99}-T}$. In Section 4.2 we show an unconditional overhead-free reduction of quantified derandomization of algorithms with two-sided error to quantified derandomization of algorithms with one-sided error. In Section 4.3 we construct a reconstructive targeted generator for quantified derandomization, and combine all the ingredients to deduce Theorem 1.6.

claim still holds when the B_j's are chosen with $O(1)$-independence, using a suitable tail bound.

4.1 Unconditional overhead-free error-reduction

Our goal in this section is to show how to reduce the error of a probabilistic time-T algorithm from $2^{-\epsilon \cdot T}$ to $2^{(T')^{.99}-T'}$, where T' is the number of coins used by the error-reduced algorithm, without significantly increasing the algorithm's runtime (i.e., the runtime overhead is near-linear, and $T' = T^{1+\epsilon}$ for a small $\epsilon > 0$).

To state the results we will need refined notions of probabilistic classes that take into account time, number of random coins, and number of exceptional ("bad") choices for random coins.

Definition 4.1. *For every time-constructible functions $Q, T, C : \mathbb{N} \to \mathbb{N}$, we say that a promise problem $\Pi = (\Pi_Y, \Pi_N)$ is in $\mathsf{prRTC}_Q[T, C]$ if there exists a randomized algorithm A that gets input $x \in \{0,1\}^n$, random bits $r \in \{0,1\}^{C(n)}$ and runs in time $T(n)$, such that*

$$\Pr_{r \in \{0,1\}^{C(n)}}[A(x;r) = 0] \leq Q(C(n))/2^{C(n)}, \qquad \text{for every } x \in \Pi_Y$$
$$\Pr_{r \in \{0,1\}^{C(n)}}[A(x;r) = 1] = 0. \qquad \text{for every } x \in \Pi_N$$

Similarly, we say $\Pi \in \mathsf{prBPTC}_Q[T, C]$ if there exists a randomized algorithm A that gets input $x \in \{0,1\}^n$, random bits $r \in \{0,1\}^{C(n)}$ and runs in time $T(n)$, such that

$$\Pr_{r \in \{0,1\}^{C(n)}}[A(x;r) = 0] \leq Q(C(n))/2^{C(n)}, \qquad \text{for every } x \in \Pi_Y$$
$$\Pr_{r \in \{0,1\}^{C(n)}}[A(x;r) = 1] \leq Q(C(n))/2^{C(n)}. \qquad \text{for every } x \in \Pi_N$$

The corresponding classes of languages are denoted by BPTC *and* RTC.

Then, the main result we prove in this section is the following.

Theorem 4.2 (overhead-free error-reduction for quantified derandomization)**.** *For every $\epsilon > 0$ and time-constructible monotone increasing function $T : \mathbb{N} \to \mathbb{N}$, we have*

$$\mathsf{prBPTIME}_{2^{(1-\epsilon)T}}[T] \subseteq \mathsf{prBPTC}_{2^{C^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C = T^{1+O(\epsilon^2)}].$$

4.1.1 Preliminaries

Let U_S denote the uniform distribution over a set S, and let U_n denote $U_{\{0,1\}^n}$. We measure the distance between distributions by their L_1 norm (i.e. statistical distance). Two distributions are *ϵ-close* if the statistical distance between them is at most ϵ.

We denote by $\mathcal{A} = \epsilon\mathcal{B} + (1-\epsilon)\mathcal{C}$ that distribution $\mathcal{A}$ is a convex combination of two independent distributions $\mathcal{B}$ and $\mathcal{C}$, where sampling $a \sim \mathcal{A}$ can be thought of as sampling $a \sim \mathcal{B}$ w.p. ϵ and sampling $a \sim \mathcal{C}$ w.p. $1-\epsilon$. We further abuse the notation of $\mathcal{B} + \mathcal{C}$ and $\mathcal{B} \cdot \mathcal{C}$ when $\mathcal{B}$ and $\mathcal{C}$ are independent distributions over some field $\mathbb{F}$. In this case, $\mathcal{A} = \mathcal{B} + \mathcal{C}$ means that a sample $a \sim \mathcal{A}$ is obtained by sampling $b \sim \mathcal{B}, c \sim \mathcal{C}$ and compute $a = b + c$, and $\mathcal{A} = \mathcal{B} \cdot \mathcal{C}$ is defined likewise. When we use '+' between

distributions to represent a convex combination, we will make the meaning clear by explicitly mentioning it in the context.

We recall standard definitions from extractor and condenser theory, which will be used throughout the section.

Definition 4.3 (Min-Entropy)**.** *The min-entropy of a random variable X, denoted by $H_\infty(X)$, is defined as the largest number b s.t. $\forall x \in \mathrm{supp}(X) : \Pr[X = x] \leq 2^{-b}$.*

Definition 4.4 ((n,b)-Source)**.** *A random variable X is an (n,b)-source if $\mathrm{supp}(X) \subseteq \{0,1\}^n$ and $H_\infty(X) \geq b$. We define the min-entropy rate of an (n,b)-source as b/n.*

Definition 4.5 (Condenser)**.** *A function $f : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $[\rho \mapsto \rho']_{\sim\gamma}$-condenser if for every $(n,\rho n)$-source X, the distribution of $f(X;U_d)$ is γ-close to some $(m,\rho' m)$-source.*

Definition 4.6 (Somewhere-(n,b)-Source)**.** *An elementary ℓ-wise somewhere-(n,b)-source is a vector of sources $(X_1,\ldots,X_\ell) \in (\{0,1\}^n)^\ell$, such that some X_i is an (n,b)-source. An ℓ-wise somewhere-(n,b)-source (or briefly somewhere-(n,b)-source, $(n,b)^{1:\ell}$-source) is a convex combination of elementary ℓ-wise somewhere-(n,b)-sources.*

Definition 4.7 (Somewhere-Condenser)**.** *A function $f : \{0,1\}^n \to (\{0,1\}^m)^\ell$ is a $[\rho \mapsto \rho']^{1:\ell}_{\sim\gamma}$-somewhere-condenser if for every $(n,\rho n)$-source X, $f(X)$ is γ-close to an $(m,\rho' m)^{1:\ell}$-source.*

To compose somewhere-condensers, we use the following analogue of [BKS+10, Lemma 5.5]. Though we are using a stronger definition of somewhere-sources, the proof in [BKS+10, Lemma 5.5 & Remark 4.9] also applies to the version below.

Lemma 4.8 (Composing Somewhere-Condensers, Extended from [BKS+10, Lemma 5.5])**.** *Let*

$$C_1 : \{0,1\}^{n_1} \to (\{0,1\}^{n_2})^{\ell_1} \quad \text{and} \quad C_2 : \{0,1\}^{n_2} \to (\{0,1\}^{n_3})^{\ell_2}$$

be a $[\rho_1 \mapsto \rho_2]^{1:\ell_1}_{\sim\gamma_1}$-somewhere-condenser, and a $[\rho_2 \mapsto \rho_3]^{1:\ell_2}_{\sim\gamma_2}$-somewhere-condenser, respectively. Denote by $C : \{0,1\}^{n_1} \to (\{0,1\}^{n_3})^{\ell_2\ell_1}$ the following composition of C_2 and C_1: for every $x \in \{0,1\}^{n_1}$ we obtain $C(x)$ by applying C_2 on each of the ℓ_1 output elements of $C_1(x)$. Then, C is a $[\rho_1 \mapsto \rho_3]^{1:\ell_1\ell_2}_{\sim\gamma_1+\gamma_2}$-somewhere-condenser.

Definition 4.9 (Merger)**.** *A function $M : (\{0,1\}^n)^\ell \times \{0,1\}^d \to \{0,1\}^m$ is a $[\rho \mapsto \rho']^{\ell:1}_{\sim\gamma}$-merger if for every $(n,\rho n)^{1:\ell}$-source $(X_1,\ldots,X_\ell)$, the distribution $f(X_1,\ldots,X_\ell;U_d)$ is γ-close to having min-entropy rate ρ'.*

4.1.2 Technical overview

As explained in Section 2.1, a standard approach to build error reduction relies on samplers, which are known to be equivalent to extractors. However, extractors require

a seed of length $\log(n-k)$, and this lower bound causes an additional multiplicative overhead of $n = T$ when enumerating seeds during error reduction of T-time algorithms. Through careful analysis, we observe that the sampling properties of condensers are useful enough for our error reduction application, where the key advantage is that condensers for our parameter setting avoid the lower bound, and may have a seed of sub-logarithmic length (see Theorem 4.15).[25] For this purpose, we need a condenser that transforms a subconstant-rate source into a constant-rate source. To minimize time overhead, the condenser should also be computed efficiently.

Our construction (see Theorem 4.14) consists of two parts: a somewhere-condenser and a merger. The somewhere-condenser is based on Zuckerman's result [Zuc06, Theorem 8.3].[26] For sources with sub-constant rate, we repeatedly compose the basic construction with itself until we reach a constant rate. The construction is in Theorem 4.13. To fill in the gap between the somewhere-condenser and an efficient condenser, we need to merge multiple (possibly correlated) sources into one source while preserving the best min-entropy rate among all input sources. Indeed, this is a merger (defined above), and a similar construction, but from a constant-entropy source rather than a sub-constant entropy one, was already used in [BDT20].

In our application, the seed length of the merger must be small (as above, to reduce the time overhead introduced by enumerating the seed), while we allow a multiplicative constant rate loss and a constant error. The is because we only need the output source to achieve a constant rate when the somewhere-random source input also has a constant rate, and the error reduction permits the condenser to have constant error. Fortunately, Dvir and Wigderson [DW08, Theorem 5.1] provided the following construction, which (given the improved analysis by Dvir, Kopparty, Saraf and Sudan [DKS+13, Theorem 18]) satisfies our requirements.

Theorem 4.10 ([DW08; DKS+13]). *For any real constants $\lambda > 0$ and $\gamma = \gamma(n) > 0$, there exists an explicit $[\rho \mapsto (1-\lambda)\rho]^{\ell:1}_{\sim\gamma}$-merger M_{DW}, with seed length*

$$d = \lambda^{-1}\log(2\ell/\gamma).$$

Moreover, the outputs of the merger on all possible seeds are computable in time $T_{\mathsf{batch\text{-}DW}}(n,\ell) = \tilde{O}(\ell^3 + n\ell^{1+1/\lambda})$.

Although the two cited works do not provide the precise time complexity of the merger, it can be directly inferred from the construction details, and we provide a proof for the time complexity in Appendix C.1.

4.1.3 The condenser construction

The basic component that we will use is a somewhere-condenser of Zuckerman [Zuc06] that increases the min-entropy rate of the input by a multiplicative constant factor, and

[25] Sampling properties of condensers were previously used for different applications, see, e.g., [DPW14; BDT19].

[26] We also propose an alternative construction based on Raz's result [Raz05, Theorem 10] in the appendix since it's of independent interest.

produces a constant number of sources in each iteration. Specifically:

Lemma 4.11 ([Zuc06, Theorem 8.3])**.** *For every constant $0 < c < 1$ there exists a constant $\alpha = O(c)$ such that for every $\delta \leq 1 - c$ and $N = 2^n$ for which $N^\delta = \omega(1)$ and n be a prime, there exists a $[\delta \mapsto (1 + \alpha/2)\delta]^{1:2}_{\sim\gamma}$-somewhere condenser $\mathsf{SRC} : [N^3] \to [N^2]^2$, where $\epsilon = N^{-\alpha\delta/20}$. Moreover, the function SRC can be computed in $O(n \log n)$ time.*

Later on we will have a constraint that N should be a prime power of 2. To deal with this constraint, we find a prime $p \in [n, (1 + \beta)n]$ using AKS primality test [AKS04] for some constant β and pad the input with zeros to length 2^{3p}. The padded version for general $N = 2^n$ is as follows.

Lemma 4.12. *For every constant $0 < c < 1$ there exists a constant $\alpha = O(c)$ such that for every $\delta \leq 1 - c$, $\beta \in (0, 1)$ and $N = 2^n$ for which $N^\delta = \omega(1)$, there exists a $[\delta \mapsto \delta \cdot (1 + \alpha/2)/(1 + \beta)]^{1:2}_{\sim\gamma}$-somewhere condenser $\mathsf{SRC} : [N^3] \to [M]^2$, where*

$$M \in [N^2, N^{2(1+\beta)}], \quad \epsilon \leq N^{-\alpha\delta/20}.$$

Moreover, the function SRC can be computed in $\tilde{O}(n)$ time.

Proof. For any fixed β and sufficiently large n, we know that there is always a prime p in $[n, (1 + \beta)n]$. We use AKS primality test to find this p, pad the input with zeros to length p, then apply Lemma 4.11 on it.

The output length of each source is $2^{2p} \in [N^2, N^{2(1+\beta)}]$, and the error is bounded by $(2^p)^{-\alpha\delta/20} \leq N^{-\alpha\delta/20}$. The rate of the padded source is at least $\delta/(1 + \beta)$, so the output source rate is at least $\delta \cdot (1 + \alpha/2)/(1 + \beta)$.

Finding p takes $\tilde{O}(n)$ time, and applying Lemma 4.11 takes $O(p \log p) = \tilde{O}(n)$ time. These sum up to $\tilde{O}(n)$ running time in total. ■

Using the basic somewhere-condenser above as a sub-component, we obtain the following somewhere-condenser. This somewhere-condenser maps n-bit inputs to m-bit sources, where $m = n^{1-\eta}$, and condenses min-entropy rate $n^{-\Omega(1)}$ to min-entropy rate $1 - \delta$.

Theorem 4.13 (Somewhere-Condenser for Subconstant-Rate Inputs)**.** *For any constants $\delta, \eta \in (0, 1)$, there exists a constant $\xi = O(\delta\eta)$ and a $[\rho \mapsto 1 - \delta]^{1:\ell}_{\sim\gamma}$-somewhere-condenser SCon for every $\rho = \rho(n) \geq n^{-\xi}$, where*

$$m = n^{1-\eta}, \quad \ell \leq n^{2\eta}, \quad \gamma = 2^{-\Theta(n^{1-\eta})}.$$

Moreover, SCon can be computed in $O(n^{1+O(\eta)})$ time.

The somewhere-condenser above is achieved by iteratively applying the basic somewhere-condenser, increasing the min-entropy rate of the input by a multiplicative constant factor and producing a constant number of sources in each iteration. Since the technical proof is relatively standard, we put it in Appendix C.1.

Composing the somewhere-condenser of Theorem 4.13 and the merger of Theorem 4.10, we get the condenser for subconstant rate input.

Theorem 4.14 (Fast Condenser for Subconstant Rate Inputs). *For any two constants $\epsilon, \gamma \in (0,1)$, there exists a constant $\xi = O(\epsilon^3)$ and a function family $\mathsf{Con} = \{\mathsf{Con}_n\}_{n\in\mathbb{N}}$ where*

$$\mathsf{Con}_n\colon \{0,1\}^n \times \{0,1\}^{d=\epsilon\cdot\log(n)+O(1)} \to \{0,1\}^{m=n^{1-\epsilon^2/4}}$$

is an $[n^{-\xi} \mapsto 1-\epsilon]_{\sim\gamma}$-condenser and the multiset of strings $\left\{\mathsf{Con}_n(x;s) : s \in \{0,1\}^d\right\}$ is computable in time $O(n^{1+O(\epsilon)})$.

The proof of Theorem 4.14 from Theorem 4.13 and Theorem 4.10 follows standard ideas (for combining a somewhere-condenser and a merger), so we postpone it too to Appendix C.1.

4.1.4 Overhead-free error-reduction using the fast condenser

We now show the error reduction result using condensers.

Theorem 4.15 (Error Reduction by Condenser). *Let $\alpha, \beta > 0$ and let T_b be sufficiently large. Let $B : \{0,1\}^{T_b} \to \{0,1\}$ such that $|B^{-1}(\sigma)| \le 2^{(1-\beta)\cdot T_b}$ for some $\sigma \in \{0,1\}$, and let*

$$\mathsf{Con} : \{0,1\}^{T_a} \times \{0,1\}^d \to \{0,1\}^{T_b}$$

be a $[T_a^{-\alpha} \mapsto 1-\beta/2]_{\sim 1/6}$-condenser. Then, the function $A : \{0,1\}^{T_a} \to \{0,1\}$ defined by

$$A(z) = \mathsf{MAJ}_{s\in\{0,1\}^d}\left\{B(\mathsf{Con}(z,s))\right\},$$

satisfies $|A^{-1}(\sigma)| \le 2^{T_a^{1-\alpha}}$.

Proof. Let $S \subseteq \{0,1\}_a^T$ be any set of size $|S| \ge 2^{T_a^{1-\alpha}}$. Since $H_\infty(U_S) \ge T_a^{1-\alpha}$, the distribution $\mathsf{Con}(U_S; U_d)$ is $1/6$-close to some distribution with min-entropy $(1-\beta/2)T_b$. Therefore, we can represent $\mathsf{Con}(U_S; U_d)$ as $\sum_{i\in[t]} c_i\mu_i + (1/6)\cdot\mu'$, where each μ_i is a flat distribution over a set of size at least $2^{(1-\beta/2)T_b}$ and μ' is an arbitrary distribution.

Since $|B^{-1}(\sigma)| \le 2^{(1-\beta)T_b}$, for every $i \in [t]$ we have

$$\Pr_{z\sim\mu_i}[B(z)=\sigma] \le \frac{|B^{-1}(\sigma)|}{|\mathrm{supp}(\mu_i)|} \le 2^{-(\beta/2)\cdot T_b},$$

and hence

$$\begin{aligned}
\Pr_{z\sim\mathsf{Con}(U_S,U_d)}[B(z)=\sigma] &= \sum_{i\in[t]} c_i \Pr_{z\sim\mu_i}[B(z)=\sigma] + \frac{1}{6}\cdot\Pr_{z\sim\mu'}[B(z)=\alpha] \\
&\le \sum_{i\in[t]} c_i 2^{-(\beta/2)\cdot T_b} + \frac{1}{6} \\
&\le 2^{-(\beta/2)\cdot T_b} + \frac{1}{6}\,.
\end{aligned}$$

By Markov's inequality,

$$\begin{aligned}
\Pr_{z\sim S}[A(z)=\sigma] &= \Pr_{z\sim S}\left[\Pr_{s\sim U_d}[B(\mathsf{Con}(z,s))=\sigma] > 1/2\right] \\
&< 2\mathbb{E}_{z\sim S}\left[\Pr_{s\sim U_d}[B(\mathsf{Con}(z,s))=\sigma]\right] \\
&= 2\Pr_{z\sim \mathsf{Con}(U_S,U_d)}[B(z)=\sigma] \\
&\leq 2^{-(\beta/2)\cdot T_b} + 1/3 \\
&\leq \frac{1}{2}\,,
\end{aligned}$$

where the last step holds for sufficiently large T_b.

That is to say, for any set $S \subseteq \{0,1\}^{T_a}$ of size larger than $2^{T_a^{1-\alpha}}$, most elements $s \in S$ satisfy $A(s) \neq \sigma$. In particular, it cannot be that $|A^{-1}(\sigma)| > 2^{T_a^{1-\alpha}}$. ■

Finally, Theorem 4.2 follows by combining Theorem 4.15 with the condenser in Theorem 4.14.

4.2 Two-sided error vs one-sided error in quantified derandomization

Next we show how to reduce quantified derandomization of algorithms with two-sided error to quantified derandomization of algorithms with one-sided error. The main point is that this reduction incurs essentially no time overhead, unlike the known reductions for general (i.e., non-quantified) derandomization, which incur either a large polynomial overhead (for black-box derandomization) or compositional overhead (for non-black-box-derandomization); see [Sip83; Lau83; ACR98; ACR+99; BF99; GZ11; GVW11; CH20; LPT24].

As explained in Section 2.1, the main observation is that in the proof of Goldreich and Zuckerman [GZ11], the overhead comes only from reducing general derandomization to quantified derandomization, and the reduction itself (i.e., the last step in their argument) is overhead-free and between quantified derandomization problems. The proof below imitates the corresponding step in their argument.

Lemma 4.16 (reducing QD with two-sided error to QD with one-sided error)**.** *For every monotone increasing time-constructible function $T, C, Q : \mathbb{N} \to \mathbb{N}$ satisfying $\forall x \in \mathbb{N} : Q(x) < 2^{x/2}$, it holds that*

$$\mathsf{prBPTC}_Q[T,C] \subseteq \mathsf{prRTC}_{Q'}[T, C/2]^{\mathsf{prcoRTC}_{Q'}[T',O(n)]},$$

where $Q'(x) = Q(2x)$ and $T'(n)$ be such that $T'(n + C(n)/2) = T(n)$. Moreover, the oracle is queried only once on an instance of length $\ell(n) = n + C(n)/2$.

For the class BPTC *of decision problems, the following variant holds*

$$\mathsf{BPTC}_Q[T,C] \subseteq \mathsf{RTC}_{Q'}[T, C/2]^{\mathsf{prcoRTC}_{Q'}[T',O(n)]},$$

and the promise problem decided by the oracle has a verifier $V(x;r)$ that either accepts or rejects at most $Q'(O(n))$ coins on every input x (even outside the promise).

Proof. Let $L = (L_Y, L_N)$ be a promise problem in $\mathsf{prBPTC}_Q[T, C]$ with a time-$T(n)$ verifier V, i.e.,

$$\Pr_{r\in\{0,1\}^C}[V(x;r) = 1] \geq 1 - Q(C)/2^C, \quad \text{for every } x \in L_Y$$
$$\Pr_{r\in\{0,1\}^C}[V(x;r) = 0] \geq 1 - Q(C)/2^C. \quad \text{for every } x \in L_N$$

We define $\Pi = (\Pi_Y, \Pi_N)$ for every $(x, r) \in \{0,1\}^n \times \{0,1\}^{C/2}$ such that

$$\Pi_Y := \left\{ (x,r) : \Pr_{y\in\{0,1\}^{C/2}}[V(x;ry) = 1] = 1 \right\},$$
$$\Pi_N := \left\{ (x,r) : \Pr_{y\in\{0,1\}^{C/2}}[V(x;ry) = 1] \leq Q(C)/2^{C/2} \right\}.$$

We note that since $Q(x) < 2^{C/2}$, the two sets Π_Y and Π_N do not intersect.

We let $Q'(x) = Q(2x)$ and $T'(n)$ such that $T'(n + C(n)/2) = T(n)$. To see that Π is in $\mathsf{prcoRTC}_{Q'}[T', O(n)]$, we consider the obvious verifier $V'(x, r; y) := V(x; ry)$ running in $T(n) = T'(n + C(n)/2)$ time, where $n + C(n)/2$ is the input size for Π. The number of coins used by V' is $C(n)/2$, which is linear in its input size as well.[27] Yes-instances in Π_Y are accepted by V' with probability 1, and no-instances in Π_N are accepted by V' with probability at most $Q(C)/2^{C/2}$. Moreover, if L is a language, then V either rejects or accepts at most $Q(C)$ coins for every input x (since $L \in \mathsf{BPTC}_Q[T, C]$). We infer that for instances $(x, r) \notin \Pi_Y \cup \Pi_N$ outside the promise, the verifier $V'(x, r; y)$ either rejects or accepts at most $Q(C) = Q'(C/2)$ coins.

We now prove that L can be reduced to Π by the randomized mapping $x \mapsto (x, r)$, where $r \in \{0,1\}^{C/2}$ is sampled uniformly at random. More concretely, we will show two claims:

1. For every $x \in L_Y$, $\Pr_r[(x, r) \in \Pi_Y] > 1 - Q(C)/2^{C/2}$.
2. For every $x \in L_N$, $\Pr_r[(x, r) \in \Pi_N] = 1$.

To prove the first claim, note that if $\mathbb{E}_y[V(x; ry)] < 1$ for some fixed r, then $\mathbb{E}_y[1 - V(x; ry)] \geq 2^{-C/2}$. By Markov's inequality and the definition of Π_Y, we know that

$$\begin{aligned}
\Pr_{r\in\{0,1\}^{C/2}}[(x,r) \notin \Pi_Y] &= \Pr_r[\mathbb{E}_y[V(x;ry)] < 1] \\
&= \Pr_r[\mathbb{E}_y[1 - V(x;ry)] \geq 2^{-C/2}] \\
&\leq \mathbb{E}_r\mathbb{E}_y[1 - V(x;ry)]/2^{-C/2} \\
&\leq Q(C)/2^{C/2}.
\end{aligned}$$

[27] In the lemma's statement, the number of random coins for the $\mathsf{prcoRTC}$ oracle is upper-bounded by $O(n)$. Indeed, if $C(n) \ll n$ this is a loose upper bound, since the input length to the oracle is $n + C/2$ and the number of random coins is $C/2$.

Next we prove the second claim: for every $x \in L_N$, by definition we know

$$\mathbb{E}_r \mathbb{E}_y[V(x; ry)] \leq Q(C)/2^C.$$

For the sake of contradiction, suppose there exists some $r_0 \in \{0,1\}^{C/2}$ such that $(x, r_0) \notin \Pi_N$, which means that

$$\mathbb{E}_y[V(x; r_0 y)] > Q(C)/2^{C/2}.$$

Therefore,

$$\mathbb{E}_r \mathbb{E}_y[V(x; ry)] \geq 2^{-C/2} \mathbb{E}_y[V(x; r_0 y)] = Q(C)/2^C,$$

which contradicts the condition $x \in L_N$. We conclude that the mapping $x \mapsto (x, r)$ maps $x \in L_N$ to an instance in Π_N with probability 1.

Hence, L can be reduced to Π by the randomized mapping $x \mapsto (x, r)$, which uses $|r| = C(|x|)/2$ random bits and runs in $C(|x|)$ time, and has one-sided error probability $Q(C)/2^{C/2}$. In other words, the problem L is in $\mathsf{prRTC}_{Q'}[C, C/2]^{\mathsf{prcoRTC}_{Q'}[T', O(n)]}$. Moreover, since the two claims above hold for every $x \in L_y \cup L_N$, if L is a language (i.e., $L \in \mathsf{BPTC}_Q[T, C]$) then the randomized reduction is correct for every input, and hence $L \in \mathsf{RTC}_{Q'}[C, C/2]^{\mathsf{prcoRTC}_{Q'}[T', O(n)]}$.[28] The claim follows since $T \geq C$ without loss of generality. ■

4.3 Quantified derandomization from uniform hardness assumptions

In this section we build a targeted hitting-set generator that allows overhead-free quantified derandomization under a uniform hardness assumption. Specifically, its correctness is based on time-space tradeoffs for randomized non-deterministic machines.

This section consists of four parts. First, in Section 4.3.1 we show how to deduce the conclusion from *non-uniform* assumptions, relying on the ingredients in previous sections and on generators from prior work. Next, in Section 4.3.2 we show the construction of a core technical component in our construction of the targeted generator (i.e., a bootstrapping system). In Section 4.3.3 we construct the targeted generator itself. Finally, in Section 4.3.4 we use the generator to deduce the quantified derandomization result under a uniform hardness assumption.

Preliminaries. For a string $S \in \Sigma^n$ and $i < j$, we denote by $S[i : j)$ the consecutive substring of S starting from index $\max(i, 0)$ and ending at index $\min(j, n) - 1$. We take the min/max operation as a part of the definition for simplicity when dealing with boundaries.

We denote by $g \circ b$ the integer obtained by concatenating the binary representations of g and b. Specifically, we only apply this operation with $b \in \{0,1\}$ so we can recover g and b from $g \circ b$ using $g = \lfloor (g \circ b)/2 \rfloor$ and $b = (g \circ b) - 2g$. For two integers a and b, we denote by $a \oplus b$ the bitwise XOR of a and b, and by $a \wedge b$ the bitwise AND of a and b.

[28] We remark that the $O(n)$ notation in the oracle denotes by n the length of the input to the *oracle*.

The upper-bound: sufficiently uniform circuits. In our uniform hardness assumption, the type of circuit families we will work with is as follows.

Definition 4.17 (sufficiently uniform circuit families)**.** *Let $d, T, \ell : \mathbb{N} \to \mathbb{N}$ be time-constructible functions such that $T(n)$ is always a power of 2. We say that $C = \{C_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ is a sufficiently uniform circuit family of width $T(n)$ and depth $d(n)$ if the following holds:*

- **Circuit structure.** *For every $n \in \mathbb{N}$, the circuit C_n has $d+1$ layers, each with T gates. The first layer contains all literals $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$, and subsequent layers consist of NAND gates with fan-in 2 and fan-out 2 (see below about removing the fan-out 2 restriction).*
- **Uniformly printing the circuit.** *The circuit description can be printed in time $\tilde{O}(d \cdot T)$, in the form of an adjacency list that maps each gate g at layer i and edge label $b \in \{0,1\}$ to g's target gate $\mathrm{OUT}_i(g,b)$ in layer $i+1$ for all $i \in [d]$.*
- **Uniformly deciding connectivity.** *The circuit family C is $\mathrm{polylog}(T)$-time-uniform, i.e., its direct connection language[29] is decidable in time $\mathrm{polylog}(T)$.*

The restriction in Definition 4.17 that gates have fan-out 2 can be removed with only a minor overhead, at the cost of assuming that the function mapping (g,i) to the name of the ith output gate of g is efficiently computable. The construction is not completely obvious (e.g., we cannot simply add a tree above each gate), due to the simultaneous restrictions on the size (i.e., we cannot afford a quadratic blow-up) and on the circuit family's uniformity. See Proposition C.4 for details and a proof.

The computational model for which we assume hardness. We will assume hardness for a computational model that we call probabilistic non-deterministic algorithms.

This is a relatively weak model of computation compared to other types of probabilistic verifiers, and in particular it is a subclass of $\mathsf{MA} \cap \mathsf{AM}$.[30] Intuitively, in this model, the verifier chooses coins and the prover sends a proof *simultaneously*, without waiting for each other, yet the requirements from each party (soundness and completeness, respectively) need to hold with respect to all choices by the other party.

Specifically, the verifier gets input x and draws randomness r. For completeness, we require a single proof that will be accepted with every randomness r (akin to perfect completeness for MA, and stricter than completeness for AM); and for soundness, with high probability over r there should be no proof causing the algorithm to err on x (akin to soundness for AM, and stricter than soundness for MA).

[29] The direct connection language $L_D(C)$ consists of all strings of the form $\langle G, a, p, z\rangle$ where G is a gate in $C_{|z|}$ and the pth gate feeding into G is a.

[30] When allowing polynomial overheads we have $\mathsf{MA} \cap \mathsf{AM} = \mathsf{MA}$, and our subclass simply equals MA; this follows using the standard trick of Babai and Moran [BM88]. However, we are interested in fine-grained time and space bounds, and we cannot afford the standard overhead from [BM88].

Remark 4.18. *In this section we will abuse notation and denote the class of probabilistic non-deterministic algorithms (as defined below) by* $\mathsf{MA} \cap \mathsf{AM}$*, even though it is formally only a subclass of* $\mathsf{MA} \cap \mathsf{AM}$*. This notational abuse is primarily for readability.*

Definition 4.19 (probabilistic non-deterministic algorithms)**.** *For a machine M and function $f\colon \{0,1\}^* \to \{0,1\}^*$, we say that M is* an $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}[T,S]$ machine consistent with f *if for every $x \in \{0,1\}^n$:*

1. **Complexity.** *For every $w, r \in \{0,1\}^{T(n)}$, the machine $M(x,w,r)$ runs in time $T(n)$ and space $S(n)$, and accesses the witness w in a read-once fashion.*
2. **Completeness (possibly vacuous).** *There exists a (possibly empty) set $W_x \subseteq \{0,1\}^{T(n)}$ of good witnesses such that $\forall w \in W_x$ it holds that $\Pr_{r \sim U_{R(n)}}[M(x,w,r) = f(x)] = 1$.*
3. **Soundness.** *With probability 2/3 over the randomness, there is no witness w causing $M(x,w,r)$ to output an incorrect answer; that is,*

$$\Pr_r[\forall w : M(x,w,r) \in \{f(x), \perp\}] \geq 2/3.$$

We may allow M to be an oracle machine, in which case for a class $\mathcal{O}$ of oracles, we will require that all three conditions hold with respect to every x and every oracle $O \in \mathcal{O}$, where the witness-set $W_x = W_{x,O}$ for every x may depend on the oracle O.

We say that a probabilistic non-deterministic algorithm computes f if completeness is not vacuous. In more detail:

Definition 4.20 (probabilistic non-deterministic algorithms computing a function)**.** *Using the same notation as Definition 4.19, we say that M* computes f at x *if $W_x \neq \emptyset$. We say that f is* hard against $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}[T,S]$ on all but finitely many input lengths *if every $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}[T,S]$ machine that is consistent with f computes f at finitely many inputs (at most).*

4.3.1 Quantified derandomization from non-uniform assumptions

We combine the error-reduction from prior sections with the following result, which is a minor adaptation of [CT21b, Theorem 5.4], to deduce overhead-free quantified derandomization from circuit lower bounds.

Theorem 4.21 (an adaptation of [CT21b, Theorem 5.4])**.** *For every $\gamma, \delta, \eta > 0$ the following holds. Assume that there exists $L \in \mathsf{DTIME}[2^n]$ such that for every $n \in \mathbb{N}$ the truth table of L can be printed in time $2^{(1+\delta)\cdot n}$, but L cannot be computed by SVN circuits of size $2^{(1-\gamma/4)\cdot n}$. Then*

$$\mathsf{prBPTC}_{2^{C^{1-\gamma}}}[C^{1+\eta}, C] \subseteq \mathsf{prDTIME}[C^{1+2\eta+2\gamma+\delta}].$$

Proof sketch. We adapt the proof in [CT21b, Theorem 5.4] to derandomize prBPTC, i.e. to the case where the number of random coins C differs from the runtime $C^{1+\eta}$. As in the original proof we use a hard truth-table of length $N^{1+\beta}$, where $N = C$ is the output

length of the generator from [CT21b, Proposition 5.2], but now we choose a slightly larger constant $\beta = 3\gamma/4 + \eta$ and keep $\alpha = \gamma/4$. The reconstruction SVN circuit size is

$$\tilde{O}(N^{1+\alpha} + C^{1+\eta} \cdot N^{\alpha} + N^{1+\beta+2\alpha-\gamma}) = \tilde{O}(C^{1+\eta+\gamma/4})$$

since the running time of D is $C^{1+\eta}$, in which case the proof by contradiction still holds. The choice of β incurs a multiplicative overhead of $C^{2\eta}$ in the deterministic simulation time. ■

Theorem 4.22 (overhead-free quantified derandomization from circuit lower bounds)**.** *For any $\epsilon > 0$, there is a constant α such that the following holds. Assume that there exists $L \in \mathsf{DTIME}[2^n]$ such that for every $n \in \mathbb{N}$ the truth table of L can be printed in time $2^{(1+\alpha)n}$, but L cannot be computed by SVN circuits of size $2^{(1-\alpha/4)n}$. Then, for $B(T) = 2^{(1-\epsilon)T}$, we have*

$$\mathsf{prBPTIME}_B[T] \subseteq \mathsf{prDTIME}[T^{1+O(\epsilon)}].$$

Proof. Applying Theorem 4.2 with parameters $\epsilon/2$ and T, we get

$$\mathsf{prBPTIME}_{2^{(1-\epsilon)T}}[T] \subseteq \mathsf{prBPTC}_{2^{C^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C = T^{1+O(\epsilon^2)}].$$

Then we apply Theorem 4.21 with $\gamma = \delta = O(\epsilon^3)$ and $\eta = O(\epsilon)$ to deduce that

$$\mathsf{prBPTC}_{2^{C^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C] \subseteq \mathsf{prDTIME}[T^{1+O(\epsilon)}],$$

choosing $\alpha = \alpha(\epsilon) = \Theta(\epsilon^3)$. ■

4.3.2 A bootstrapping system with block locality

We construct a bootstrapping system a-la [CT21a], which is an encoding of a circuit that preserves downward self-reducibility while adding additional useful features. The crucial point in our construction is that the bootstrapping system we build has a locality property, which means that every substring in a row can be calculated by querying constantly many substrings of the same length in the previous row. As explained in Section 2.1, this property will be crucial for the reconstruction algorithm of our targeted HSG.

Theorem 4.23 (bootstrapping system with block locality)**.** *Let $d, T, \ell : \mathbb{N} \to \mathbb{N}$ be time-constructible functions such that $T(n)$ is always a power of 2 and $T(n) \geq n$. Let $C = \{C_n : \{0,1\}^n \to \{0,1\}^{\ell(n)}\}_{n\in\mathbb{N}}$ be a sufficiently uniform circuit family of width $T(n)$, depth $d(n)$ and output length ℓ. Let $f : \{0,1\}^* \to \{0,1\}^*$ be the function computed by C. Then, for every $x \in \{0,1\}^*$ there exists a sequence of strings $B_1(x), ..., B_{d'+1}(x) \in [A]^{T'}$ with the following properties, where alphabet size $A = 2T$, sequence length $d' = O(d \cdot \log^2 T)$, and string length $T' = 2T$:*

1. *(**Input Layer.**) There exists an algorithm $A_{\text{in}}(x, j)$ that gets input $(x, j) \in \{0,1\}^n \times \{0, 1, \ldots, T' - 1\}$ and outputs $B_1(x)_j$ in time $O(\log T)$.*

2. *(**Downward self-reducibility.**) There exists an nondeterministic algorithm $A_{\mathrm{DSR}}(x, i, j; w)$ that gets input $(x, i, j) \in \{0,1\}^n \times \{2, 3, \ldots, d'+1\} \times \{0, 1, \ldots, T'-1\}$, witness string $w \in \{0,1\}^{\log T}$ and oracle access to $B_{i-1}(x)$, and satisfies the following for every input (x, i, j):*

   - *For every $w \in \{0,1\}^{\log T}$, the output $A_{\mathrm{DSR}}(x, i, j; w) \in \{\perp, B_i(x)_j\}$;*
   - *The set $\{w : A_{\mathrm{DSR}}(x, i, j; w) = B_i(x)_j\}$ of good witnesses is nonempty.*

   This algorithm runs in time $\mathrm{polylog}(T)$.

3. *(**Batch-computability.**) There exists a pair of deterministic algorithms $(A_{\mathrm{prep}}, A_{\mathrm{batch}})$ such that:*

   - *The preprocessing algorithm A_{prep} gets input 1^n and outputs an advice string $\mathrm{adv}_n \in \{0,1\}^{\tilde{O}(d \cdot T)}$ in time $\tilde{O}(d \cdot T)$.*
   - *The batch-computing algorithm A_{batch} gets input $(1^n, i) \in \{1^n\} \times [d'+1]$, the advice string adv_n, and oracle access to $B_{i-1}(x)$, and outputs the entire layer $B_i(x)$ in time $\tilde{O}(T)$.*

4. *(**Output Layer.**) For every $i \in [n]$, the output layer $B_{d'+1}(x)$ satisfies $B_{d'+1}(x)_{2(i-1)} = B_{d'+1}(x)_{2(i-1)+1} = f(x)_i$.*

5. *(**Block Locality.**) For every $s \in \mathbb{N}$, if we partition every $B_i(x)$ into blocks of size s (i.e., for $k \in \{0, 1, \ldots, \lceil T'/s \rceil - 1\}$, the block k of $B_i(x)$ is $B_i(x)[ks : (k+1)s)$), then the following holds. For every $2 \le i \le d'+1$ and every block in $B_i(x)$, when we apply the downward self-reducibility algorithm from Item 2 to compute all of the entries in the block, the set of all the positions in $B_{i-1}(x)$ that are queried is covered by at most 5 blocks in $B_{i-1}(x)$.*

We call such a sequence of strings a $(d' \times T')$ bootstrapping system.

For simplicity we omit the (x) notation when we refer to $B_i(x)$ in this section.

Proof. We first describe the construction of the bootstrapping system, then show that it has the claimed properties.

Construction. Our bootstrapping system consists of 3 types of strings, which are also called layers:

1. **Original Layers.** For layer $i \in [d+1]$ in C_n we build an *original layer* $P_i \in \{0,1\}^{2T}$ by replicating each gate g in the ith layer of C_n twice in P_i and indexing the two copies by $g \circ 0$ and $g \circ 1$ respectively, i.e., $(P_i)_{g \circ b}$ will indicate the value of gate g in the ith layer of C_n (see Lemma 4.25).

2. **Translation Layers.** For $i \in [d]$, we set up a *translation layer* Q_i based on P_i with gates indexed by $g \circ b$ for $g \in \{0, 1, \ldots, T-1\}$ and $b \in \{0, 1\}$, such that $(Q_i)_{g \circ b} := (\mathrm{OUT}_i(g, b), (P_i)_{g \circ b}) \in \{0, 1, \ldots, T-1\} \times \{0, 1\}$ where $\mathrm{OUT}_i(g, b)$ is the bth gate in C_n receiving the output of the gth gate in the ith layer of C_n.

   We note that Q_i contains sufficient information to construct P_{i+1}, including which value each gate in P_i outputs and which gate in P_{i+1} will take this value as input.

3. **Auxiliary Layers.** For $i \in [d]$, we use a series of *auxiliary layers* $\{R_{i,k,j}\}_{k \in [\log 2T], j \in \{0,1,\ldots,k-1\}}$ to implement a bitonic sorter for Q_i, where each $R_{i,k,j}$ is a string in $(\{0, 1, \ldots, T-1\} \times \{0, 1\})^{2T}$.

These layers are structured in the bootstrapping system in the following way:

- P_i feeds its output into Q_i;
- Q_i feeds its output into $R_{i,1,0}$;
- $R_{i,k,0}$ feeds its output into $R_{i,k+1,k}$, if $k \neq \log 2T$;
- $R_{i,k,j}$ feeds its output into $R_{i,k,j-1}$, if $j \neq 0$;
- $R_{i,\log 2T,0}$ feeds its output into P_{i+1}.

And these layers work as follows:

- **Original Layers.** P_1 is just the (replicated) input layer of C_n. For $i \in \{2, 3, \ldots, d+1\}$, each gate $g \circ b$ in P_i receives inputs from gates $g \circ 0$ and $g \circ 1$ in $R_{i-1,\log 2T,0}$, whose values are two pairs $(g'_0, v_0), (g'_1, v_1) \in \{0, 1, \ldots, T-1\} \times \{0, 1\}$ respectively. We compute the NAND of v_0 and v_1 as the output value of gate $g \circ b$ in P_i.
- **Translation Layers.** For $i \in \{2, 3, \ldots, d+1\}$, each gate $g \circ b$ in Q_i receives input from gate $g \circ b$ in P_i and computes the function $v \mapsto (\mathrm{OUT}_i(g, b), v)$.
- **Auxiliary Layers.** For each auxiliary layer $R_{i,k,j}$, each gate g in $R_{i,k,j}$ receives inputs from gates g and $g \oplus 2^j$ in the layer below, and computes the function

$$(v_0, v_1) \mapsto \begin{cases} \min(v_0, v_1), & \text{if } g \wedge 2^k = 0 \text{ and } g < g \oplus 2^j \\ \max(v_0, v_1), & \text{if } g \wedge 2^k = 0 \text{ and } g > g \oplus 2^j \\ \max(v_0, v_1), & \text{if } g \wedge 2^k \neq 0 \text{ and } g < g \oplus 2^j \\ \min(v_0, v_1), & \text{if } g \wedge 2^k \neq 0 \text{ and } g > g \oplus 2^j \end{cases} \tag{1}$$

  where we note that $v_0, v_1 \in \{0, 1, \ldots, T-1\} \times \{0, 1\}$ are pairs that we order by first comparing their first component and then comparing the second.

We now verify the properties stated in Theorem 4.23. First of all, the number of layers in the bootstrapping system is $d' := 2 \times (d+1) + d \cdot O(\log^2 T) = O(d \cdot \log^2 T)$, the length of each layer is $T' := 2T$, and the maximum alphabet size used among layers is $A = 2T$.

Algorithm for Input Layer. The algorithm $A_{\text{in}}(x, j)$ outputs $x_{\lfloor j/2 \rfloor}$ if $j < 2n$ and outputs 0 otherwise. Its correctness follows from the fact that B_1 is exactly P_1, where $(P_1)_{i \circ 0} = (P_1)_{i \circ 1} = x_i$ for $i < n$ and $(P_1)_{i \circ b} = 0$ if $i \geq n$. Its running time is $O(\log T)$.

Algorithm for Downward self-reducibility. The downward self-reducibility algorithm $A_{\text{DSR}}(x, i, j; w)$ that computes $B_i(x)_j$ with oracle access to $B_{i-1}(x)$ is implicitly described in the construction part:

- If B_i is an original layer, then we let $g = \lfloor j/2 \rfloor$, query $B_{i-1}(x)_{g \circ 0}$ and $B_{i-1}(x)_{g \circ 1}$ and output the NAND of their second component. Recall that in this case B_{i-1} is an auxiliary layer and each of its entries is a pair in $\{0, 1, \ldots, T-1\} \times \{0, 1\}$. The running time is bounded by $O(\log T)$, including the cost of querying two entries in B_{i-1} and computing the NAND of two Boolean values.
- If B_i is a translation layer, then we let $g = \lfloor j/2 \rfloor$ and $b = j - 2g$, interpret witness w as the label of gate g', and verify that $\langle g, g', b, 1^n \rangle \in L_D(C)$. If yes, we query $B_{i-1}(x)_j$ and output $(g'_b, B_{i-1}(x)_j)$; Otherwise we output $\perp$. The running time is $\text{polylog}(T)$, including the cost of 1) querying one entry in B_{i-1}, 2) computing the OUT_i function using nondeterminism and 3) the $\text{polylog}(T)$-time RAM deciding direct connectivity in C_n.
- If B_i is an auxiliary layer $R_{i',k',j'}$, then we let $k = j \oplus 2^{j'}$, query $B_{i-1}(x)_j$ and $B_{i-1}(x)_k$ and compute the function in eq. (1) on $B_{i-1}(x)_j$ and $B_{i-1}(x)_k$. The running time is $O(\log T)$, including the cost of querying two entries in B_{i-1} and computing a function that is either min or max over two $\log 2T$-bit inputs.

Therefore the running time of A_{DSR} is bounded by $\text{polylog}(T)$.

Algorithm for Batch-computability. We formally describe the two-stage batch-computability property in below.

For A_{prep}, given the input length parameter 1^n, the algorithm explicitly prints the complete adjacency list circuit description of C_n, which has size $\tilde{O}(d \cdot T)$. By Definition 4.17, this advice string adv_n is output in $\tilde{O}(d \cdot T)$ time.

For A_{batch}, given the layer index $i \in [d']$, the precomputed advice adv_n, and oracle access to $B_{i-1}(x)$, the batch-computing algorithm A_{batch} computes the entire string $B_i(x)$ as follows:

- If B_i is a translation layer $Q_{i'}$, for all indices $j = 2g + b < 2T$, the algorithm needs to compute $(Q_{i'})_{g \circ b} := (\text{OUT}_{i'}(g, b), (P_{i'})_{g \circ b})$. The value $(P_{i'})_{g \circ b}$ is obtained via a single query to the oracle $B_{i-1}(x)_j$ (since B_{i-1} is the original layer $P_{i'}$). The target gate $\text{OUT}_{i'}(g, b)$ is looked up in $O(\log T)$ time by querying the precomputed adjacency lists in adv_n. Thus, iterating over all $2T$ entries takes $O(T \log T) = \tilde{O}(T)$ time.

- Otherwise, B_i is either an original layer or an auxiliary layer. In both cases, the way in which B_i's entries are calculated from the entries of the previous layer is independent of the original wiring in C_n. Therefore, the algorithm computes each entry j deterministically by executing the same local evaluations as specified in A_{DSR}, requiring 2 oracle queries to $B_{i-1}(x)$. The evaluation takes $O(\log T)$ time for each entry (performing a NAND/min/max operation). Iterating over all $2T$ entries takes $O(T \log T) = \tilde{O}(T)$ time.

Consequently, each call of $A_{\mathrm{batch}}(i, \mathrm{adv}_n)$ runs in deterministic time $\tilde{O}(T)$.

Property of Output Layer. To prove that the property is correct, we show that the original layers in the bootstrapping system simulate C_n faithfully. The main technical lemma we need is as follows:

Proposition 4.24. *For every $i \in [d+1]$, the auxiliary layers $\{R_{i,k,j}\}_{k \in [\log 2T], j \in \{0,1,\ldots,k-1\}}$ sort their inputs in ascending order, i.e., $R_{i,\log 2T,0}$ is the sorted version of Q_i.*

Proof. To verify this, we compare the construction and the following standard pseudocode of bitonic sort. The innermost for loop with outer loop variables k and j is simulated by layer $R_{i,k,j}$, and the structure order of the layers matches the order of the loops.

Algorithm 1 Bitonic Sorter

Input: $A_0 \ldots A_{2T-1}$
Output: A sorted version of A

```
1: function BitonicSort(A[ ])
2:     for k ← 1 to log 2T do
3:         for j ← k − 1 downto 0 do
4:             for i ← 0 to 2T − 1 do
5:                 ℓ ← i ⊕ 2^j
6:                 if ℓ > i then
7:                     if (i ∧ 2^k = 0 and A[i] > A[ℓ]) or (i ∧ 2^k ≠ 0 and A[i] < A[ℓ]) then
8:                         swap A[i] and A[ℓ]
```

Therefore, $R_{i,k,j}$ is the same as the array we get after executing the corresponding iteration of the for loop, and the ultimate output $R_{i,\log 2T,0}$ is the sorted version of the input. ∎

With this lemma in hand, we can assert the behavior of the original layers.

Lemma 4.25. *For every $i \in [d+1]$, $g \in \{0,1,\ldots,T-1\}$ and $b \in \{0,1\}$, the value of $(P_i)_{g \circ b}$ is the same as the value of gate g in the ith layer of C_n.*

Proof. We prove this by showing it inductively for each $i \in [d+1]$.

For the base case where $i = 1$, this holds by the definition of P_1.

For the inductive step, we assume that the proposition holds for i and prove it for $i+1$. We remark that C is a circuit of fan-in 2 and fan-out 2, thus for each gate g in the $(i+1)^{\text{th}}$ layer of C_n there will be exactly two gates $g_0 \circ b_0$ and $g_1 \circ b_1$ in P_i such that $\text{OUT}_i(g_j, b_j) = g$ for $j \in \{0,1\}$. Therefore, $R_{i,\log T,0}$, the sorted version of the sequence $Q_i = \{(\text{OUT}_i(g,b), (P_i)_{g\circ b})\}_{g,b}$, has the following form

$$(0, v_{0\circ 0}), (0, v_{0\circ 1}), (1, v_{1\circ 0}), (1, v_{1\circ 1}), \ldots, (T-1, v_{(T-1)\circ 0}), (T-1, v_{(T-1)\circ 1}),$$

where one pair $(g, v_{g\circ b})$ means that gate g in the $(i+1)^{\text{th}}$ layer of C_n takes $v_{g\circ b}$ as one of its inputs. Here we do not distinguish between the values from P_i and from the ith layer of C_n by the inductive hypothesis. According to our construction, gate $g \circ b$ in P_{i+1} also takes $v_{g\circ 0}$ and $v_{g\circ 1}$ as input values. By the evaluation rules of P_{i+1}, we know that $(P_{i+1})_{g\circ b}$ receives the same inputs as gate g in the $(i+1)^{\text{th}}$ layer of C_n and also computes the NAND function, thus they yield the same value. ∎

Therefore the last layer P_{d+1} is a replicated version of the output layer of C_n, from which we can get $f(x)_i$ by querying either $(P_{d+1})_{(i-1)\circ 0}$ or $(P_{d+1})_{(i-1)\circ 1}$.

Block Locality. We verify the block locality property as the last step. For each layer B_{m+1} where $m \in [d']$, we do the following case analysis based on which type the layer B_{m+1} belongs to:

- B_{m+1} is an original layer P_i. In this case, the DSR algorithm that computes gate $g \circ b$ in P_i queries gates $g \circ 0$ and $g \circ 1$ in B_m, which means that the queried indexes fall in $[g \circ b - 1, g \circ b + 1]$. Thus, if we apply the DSR algorithm to recover block k in P_i (i.e., $P_i[ks : (k+1)s)$), all the queried indexes fall in range $B_m[ks-1, (k+1)s+1)$, which can be covered by at most 3 blocks in B_m: block $k-1$, block k and block $k+1$ (if they exist).

- B_{m+1} is a translation layer Q_i. In this case, the DSR algorithm that computes gate $g \circ b$ in Q_i only queries one gate $g \circ b$ in B_m. So all the queried indexes when recovering block k in Q_i (i.e., $Q_i[ks : (k+1)s)$) fall in range $B_m[ks : (k+1)s)$, which is covered by block k in B_m.

- B_{m+1} is an auxiliary layer $R_{i,k,j}$. In this case, the DSR algorithm that computes gate g in $R_{i,k,j}$ queries gates g and $g \oplus 2^j$ in B_m. Note that since 2^j is a power of 2, we have $\{g, g \oplus 2^j\} \subseteq \{g - 2^j, g, g + 2^j\}$. Therefore, the set of all the queried indexes when recovering block k in $R_{i,k,j}$ is a subset of the union of the three ranges[31]: $B_m[ks - 2^j, (k+1)s - 2^j)$, $B_m[ks, (k+1)s)$, and $B_m[ks + 2^j, (k+1)s + 2^j)$. The range $B_m[ks, (k+1)s)$ is covered by block k in B_m; each of the other two

[31]Precisely speaking, we are talking about the parts intersecting with $B_m[0, 2T-1)$ in each range, recalling the definition of the $S[i:j)$ notation.

ranges in consecutive and has size at most s, which implies that each of them can be covered by at most 2 adjacent blocks in B_m. These sum up to at most 5 blocks in total.

We conclude that one block can be reconstructed using the DSR algorithm where all the queried indexes are covered by at most 5 blocks in the previous layer. ∎

4.3.3 Targeted hitting-set generator for quantified derandomization

In this section we construct the targeted hitting-set generator that Theorem 1.6 relies on and that was described in Section 2.1. The generator is based on a hard function f computable by circuits of width W_f and depth d_f. It is parameterized by $\beta > 0$, which determines the output length $m = W^{1-\beta}$; and by $\epsilon > 0$, which determines the number of exceptional inputs $Q(m) = 2^{m^{1-\epsilon}}$ to distinguisher.

Proposition 4.26 (a reconstructive tarHSG for quantified derandomization)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be computable by a sufficiently uniform circuit family of width $W_f(n)$ and depth $d_f(n)$. For every constant $\epsilon, \beta \in (0,1)$ and time-constructible $Q : \mathbb{N} \to \mathbb{N}$ such that $n \leq Q(n) \leq 2^{n^{1-\epsilon}}$, there exist a deterministic algorithm H_f and a probabilistic oracle machine* Rec *such that for every $x \in \{0,1\}^n$:*

1. ***Generator.*** *The generator H_f gets input x and outputs a list of m-bit strings of size $\tilde{O}(d_f \cdot W_f^\beta)$, where $m = W_f^{1-\beta}$. The generator H_f runs in $\tilde{O}(d_f \cdot W_f)$ time.*

2. ***Reconstruction.*** *Let $\mathcal{D}_\sigma$ be the class of functions $D\colon \{0,1\}^m \to \{0,1\}$ such that $|D^{-1}(\sigma)| \leq Q(m)$. With respect to the class $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1$ of oracles,* Rec *is an $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}[T,S]$ oracle machine consistent with f; and for every x, if $D_x \in \mathcal{D}_0$ rejects all the strings that $H_f(x)$ prints, then Rec^{D_x} computes f at x.*

   On input x, the reconstruction Rec *runs in time T and space S, where*

$$T(n) = \tilde{O}(d_f \cdot W_f),$$
$$S(n) = \tilde{O}(W_f^{1-\beta} + W_f^\beta \cdot \log Q(m)) \leq \tilde{O}(W_f^{1-\beta} + W_f^{1-\epsilon+\epsilon\beta}),$$

   uses $O(m)$ bits of randomness and makes $\tilde{O}(d_f \cdot W_f^\beta)$ oracle queries.

Proof. We initiate a $(d' \times T')$ bootstrapping system for f from Theorem 4.23, with parameters $d' = O(d_f \cdot \log^2 W_f)$, output length $\ell(n) = n$, width $T' = O(W_f)$, and $A = T'$.[32] We get the corresponding algorithms $A_{\text{in}}, A_{\text{prep}}, A_{\text{batch}}$ and A_{DSR}.

For $k \in [d']$, we partition the kth layer $B_k \in [A]^{T'}$ in the bootstrapping system into disjoint and consecutive blocks of size $m' = m/\log(A)$; that is, $B_k = (b_{k,1}, \ldots, b_{k,t})$, where $t = T'/m' = \tilde{O}(W_f^\beta)$. Each block $b_{k,i} \in [A]^{m'}$ is also identified with $\{0,1\}^m$.

[32] We can assume that W_f is a power of two without loss of generality (i.e., by adding dummy gates).

The generator H_f. Given input $x \in \{0,1\}^n$, the generator computes the layers of the bootstrapping system $B_1, \ldots, B_{d'+1}$ and outputs the set $\{b_{k,i} : k \in [d'+1], i \in [t]\}$ of all blocks.

More precisely, H_f first uses A_{in} to compute B_1 from the input in time $\tilde{O}(T') = \tilde{O}(W_f)$, then invokes the preprocessing algorithm $A_{\text{prep}}(1^n)$, which runs in time $\tilde{O}(d' \cdot T') = \tilde{O}(d_f \cdot W_f)$ and outputs an advice string adv_n. Using adv_n, the generator H_f sequentially computes the layers $B_1, \ldots, B_{d'+1}$ of the bootstrapping system: For $k = 1, ..., d'$, it invokes the batch-computability algorithm $A_{\text{batch}}(1^n, k+1; \text{adv}_n)$, giving it oracle access to B_k, and obtains B_{k+1}. This takes time $\tilde{O}(T') = \tilde{O}(W_f)$ time per layer, so overall $\tilde{O}(d' \cdot T') = \tilde{O}(d_f \cdot W_f)$.

Finally, H_f partitions each layer into T'/m' blocks and prints all blocks across all layers, which yields an output set of size $d' \cdot t = \tilde{O}(d_f \cdot W_f^{\beta})$. This step takes time linear in the total size of the system, which is $O(d' \cdot T') = \tilde{O}(d_f \cdot W_f)$. Summing up all the costs, the running time of H_f is bounded by $\tilde{O}(d_f \cdot W_f)$.

Reconstructing f. The reconstruction will use a standard construction of an almost pairwise-independent hash function, as follows:

Lemma 4.27. *For every given $n, m \in \mathbb{N}$ and $0 < \epsilon < 1$, there exists an ϵ-almost pairwise independent hash family $\mathcal{H} = \{h_k : \{0,1\}^n \to \{0,1\}^m\}_{k \in \mathcal{K}}$ with key space $\mathcal{K} = \{0,1\}^{O(m + \log n + \log(1/\epsilon))}$ satisfying*

$$\Pr_{k \leftarrow \mathcal{K}}[h_k(x_1) = y_1 \wedge h_k(x_2) = y_2] \leq \frac{1+\epsilon}{2^{2m}}$$

for all distinct $x_1, x_2 \in \{0,1\}^n$ and $y_1, y_2 \in \{0,1\}^m$. We can sample a function h_k from $\mathcal{H}$ by sampling a key k using $O(m + \log n + \log(1/\epsilon))$ random bits. There exists a deterministic algorithm such that given $k \in \mathcal{K}$ and $x \in \{0,1\}^n$ as input, runs in time $\tilde{O}(|k| + |x|)$ and outputs $h_k(x)$.

The reconstruction procedure Rec is an oracle machine that receives input $x \in \{0,1\}^n$, random bits $r \in \{0,1\}^*$, witness $w \in \{0,1\}^*$, and oracle access to some $D \in \mathcal{D}$. (The length of r and w will be spelled out later.) The procedure does the following steps.

1. Sample $r_1 \in \{0,1\}^{m/2}$ using random bits, and guess $w_1 \in \{0,1\}^{m/2}$ using nondeterminism. Query the oracle on $D(r_1 \circ w_1)$, and return $\perp$ if the result is 0.
2. Sample a 0.1-almost pairwise independent hash function $h_r : \{0,1\}^m \to \{0,1\}^{\ell}$ from $\mathcal{H}$ using random bits r_2, where $\ell = 2 \log Q(m) + O(1)$.
3. Reserve $2t$ pieces of ℓ-bit space on its working tape as registers $v_1, \ldots, v_t, v'_1, \ldots, v'_t$, and six pieces of m-bit space as registers $g_1, \ldots, g_6$.

4. For each $i \in [t]$, compute $b_{1,i}$ by calling $A_{\text{in}}(x, j)$ for each j in this block and concatenating the result bits, store it in g_1, then assign $v_i \leftarrow h_r(g_1)$. Make an oracle query to $D(g_1)$ and if $D(g_1) \neq 0$, output $\perp$ and halt.

5. For each $k \in [d']$, do the following:

   (a) For each $i \in [t]$, compute $b_{k+1,i}$ by calling $A_{\text{DSR}}(x, k+1, j; w'_{k+1,j})$ for each j in this block and concatenating the result bits, store it in g_1, then assign $v'_i \leftarrow h_r(g_1)$. The witness $w'_{k+1,j}$ provided to the DSR algorithm is taken from a corresponding part in the witness w of Rec, and the oracle access to B_k in $A_{\text{DSR}}(x, k+1, j; w)$ is simulated using step 5b. Make an oracle query to $D(g_1)$ and if $D(g_1) \neq 0$, output $\perp$ and halt.

   (b) For each oracle query made to $B_k(x)_j$ for some j, if the query position j belongs to a block $b_{k,i'}$ that is not stored in any of $g_2, \ldots, g_6$ yet, compute $b_{k,i'}$ using step 5c and store it in the earliest updated register among $g_2, \ldots, g_6$. Otherwise query the value of $B_k(x)_j$ from the corresponding register and return the value in the corresponding register.

   (c) To compute $b_{k,i'}$, consume m bits of witness w and denote them as $s_{k,i'}$. Make an oracle query to $D(s_{k,i'})$ and evaluate $h_r(s_{k,i'})$. If $D(s_{k,i'}) = 0$ and $h_r(s_{k,i'}) = v_{i'}$, return $s_{k,i'}$ as the value of $b_{k,i'}$. Otherwise output $\perp$ and halt.

   (d) For each $i \in [t]$, assign $v_i \leftarrow v'_i$.

6. For each $i \in [n]$, call step 5b to query $B_{d+1}(x)_{2(i-1)}$. Concatenate the result bits and output it.

Correctness. We claim that with respect to any $D \in \mathcal{D}_0 \cup \mathcal{D}_1$, Rec^D is an $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}$ oracle machine consistent with f, and that Rec^D computes f at x if $D \in \mathcal{D}_0$ rejects all the strings that $H_f(x)$ prints.

We define a possibly empty set of witnesses W_x for Rec^D on input x as follows. A witness w is in W_x if:

1. its substring w_1 satisfies $\Pr_r[D(r \circ w_1) = 1] = 1$;

2. its substrings $w'_{k,j}$ for $2 \leq k \leq d+1$ and $j \in [T]$ are all good witnesses for the corresponding invocation of A_{DSR};

3. its substrings $s_{k,i}$ for $1 \leq k \leq d$ and $i \in [t]$ are all equal to the corresponding blocks $b_{k,i}$ in the bootstrapping system.

We note that W_x is nonempty if the first condition is satisfied.

For an oracle $D \in \mathcal{D}_1$, we show that the soundness of Rec^D is guaranteed, while the completeness condition is defined to have an empty good witness set. Since $|D^{-1}(1)| \leq Q(m) \leq 2^{m^{1-\epsilon}} \ll 2^{m/2}$, we know that with high probability over the choice of r_1 there is no witness w_1 satisfying $D(r_1 \circ w_1) = 1$, in which case Rec^D outputs $\perp$. Therefore the soundness condition holds, and Rec is consistent with f when $D \in \mathcal{D}_1$.

For an oracle $D \in \mathcal{D}_0$, we show that Rec^D always pass the test in step 1. Since $|D^{-1}(0)| \leq Q(m) \leq 2^{m^{1-\epsilon}} \ll 2^{m/2}$, we know that there exists a good witness w_1 satisfying $D(r_1 \circ w_1) = 1$ for every r_1, in which case Rec^D continues to run step 2-6. Moreover, this implies that W_x is nonempty for $D \in \mathcal{D}_0$.

We do the following case analysis for the rest of the steps of the procedure.

Case 1. For an oracle $D \in \mathcal{D}_0$ that rejects all the strings that $H_f(x)$ prints (i.e., D rejects all $b_{k,i}$), we define the good witness set to be W_x.

Case 1a. For a good witness $w \in W_x$, we show that Rec^D outputs $f(x)$ with probability 1. We prove that Rec is simulating the bootstrapping system of f. If Rec computes all the blocks $b_{k,i}$ correctly in step 5 for all $k \in [d'+1]$ and $i \in [t]$, then its output is $f(x)$ according to Theorem 4.23 Item 4.

Claim 4.28. *In step 5, $v_i = h_r(b_{k,i})$ for all $k \in [d']$ and $i \in [t]$ at the start of each iteration.*

Proof. The base case $k = 1$ holds by step 4 and the correctness of A_{in} (Theorem 4.23 Item 1).

For the inductive case $2 \leq k \leq d'$, by the inductive hypothesis, we have $v_i = h_r(b_{k,i})$ for all $i \in [T^\beta]$ at the start of the kth iteration. Since the witness w is good, the check in step 5c will be always successful since D rejects $s_{k,i} = b_{k,i}$ and $h_r(s_{k,i}) = h_r(b_{k,i}) = v_i$. Each call to step 5c returns the correct block $b_{k,i}$, so step 5b provides correct oracle access to B_k for A_{DSR}. As the witness $w'_{k+1,j}$ is good for $A_{\text{DSR}}(x, k+1, j; w'_{k+1,j})$ on every j in block i, the invocations build the correct value of $b_{k+1,i}$ for all $i \in [t]$ according to the correctness of A_{DSR}.

Therefore, $v'_i = h_r(b_{k,i})$ after step 5a, and $v_i = h_r(b_{k,i})$ after being assigned $v_i \leftarrow v'_i$ in step 5d. ∎

Case 1b. We show that with probability greater than $2/3$, for every witness w, the output of Rec^D falls in $\{f(x), \perp\}$.

For large enough m, the probability that the hash function h_r collides on the set of strings that D rejects is

$$\Pr_{h_r}\left[\exists s_1, s_2 \in D^{-1}(0), s_1 \neq s_2 : h_r(s_1) = h_r(s_2)\right] \leq |D^{-1}(0)|^2 \cdot \frac{1+0.1}{2^\ell} < 1/3,$$

i.e., with probability at least $2/3$, h_r does not collide on the set of strings that D rejects. We show that for these good hash functions, the output of Rec^D falls in $\{f(x), \perp\}$ for all witnesses.

If the witness w is in W_x, then it is proved in Case 1a that the output is $f(x)$.

For $w \notin W_x$, we consider the first component of w that does not match the definition of a good witness. If this component is $w'_{k,j}$ for some k, j, then the invocation of $A_{\text{DSR}}(x, k+1, j; w'_{k+1,j})$ will output $\perp$ by the correctness of A_{DSR} (Theorem 4.23 Item 2). If this component is $s_{k,i}$ for some k, i, then in the corresponding step 5c, if $D(s_{k,i}) \neq 0$, Rec^D outputs $\perp$ immediately; otherwise we know $D(s_{k,i}) = 0$ and $s_{k,i} \neq b_{k,i}$, which

implies that $h_r(s_{k,i}) \neq h_r(b_{k,i}) = v_i$ and Rec^D outputs $\perp$, since h_r does not collide on $D^{-1}(0)$.

Case 2. For an oracle $D \in \mathcal{D}_0$ that accepts some $b_{k,i}$, we define the good witness set to be empty. But we still do a case analysis here.

Case 2a. For a witness $w \in W_x$, we show that Rec^D outputs $\perp$ with probability 1. We consider the first query to D in the execution of Rec where D behaves differently from an oracle D' that rejects all the strings that $H_f(x)$ prints, suppose the input is $b_{k,i}$. Before this point, the execution of Rec^D is identical to that of $\mathsf{Rec}^{D'}$, thus the values of v_i are all equal to $h_r(b_{k,i})$ for all $k \in [d']$ and $i \in [t]$ at the start of each iteration in step 4, and the value of $b_{k,i}$ sent to D are the same as that sent to D'. Since $D(b_{k,i}) \neq D'(b_{k,i}) = 0$, the procedure immediately outputs $\perp$ and halts in the corresponding check. Moreover, observing that all the blocks $b_{k,i}$ will be sent to D in Rec, there must be a query to D returning accept. We conclude that Rec^D outputs $\perp$ with probability 1 in this case.

Case 2b. We show that with probability at least $2/3$, for every witness Rec^D outputs either $f(x)$ or $\perp$. Similarly to Case 1b, we also focus on the particular event that h_r does not collide on $D^{-1}(0)$, which has probability greater than $2/3$.

If $w \in W_x$, then Case 2a showed that Rec^D outputs $\perp \in \{f(x), \perp\}$.

For $w \notin W_x$, consider the first execution step where the procedure meets either a bad witness or an oracle query on which D behaves differently from an oracle D' that rejects all the strings that $H_f(x)$ prints. If this step is an oracle query, then the analysis is the same as Case 2a; otherwise, there is a bad witness, then the analysis is the same as Case 1b.

Complexity. We claim that Rec runs in time $T = \tilde{O}(d_f \cdot W_f)$ and space $S = \tilde{O}(W_f^{1-\beta} + W_f^{\beta} \cdot \log Q(m))$, uses $O(m)$ random bits, and makes $O(d_f \cdot W_f^{\beta})$ oracle queries.

Randomness. The number of random bits used equals $|r_1| = m/2$ plus the key length $|r_2| = O(\ell + \log m) = O(\log Q(m))$ of the hash function h_r from Lemma 4.27, which is $O(m)$ in total.

Space. The algorithm uses $O(\log T') = O(\log W_f)$ bits of space for counters, labels and temporary variables, reserves $2t \cdot \ell + 6 \cdot m = \tilde{O}(W_f^{1-\beta} + W_f^{\beta} \cdot \log Q(m))$ bits of space in step 3, evaluates h_r using $\tilde{O}(m + \ell) = \tilde{O}(m)$ space, and invokes A_{in} and A_{DSR}. These two algorithms both run in polylog(T) time according to Theorem 4.23, therefore the space used by them is bounded by polylog(T). Summing up all above, the total space used is bounded by $\tilde{O}(W_f^{1-\beta} + W_f^{\beta} \cdot \log Q(m))$.

Time. For time bound, we analyze the procedure step by step.

1. Step 1 takes linear time in m and uses an oracle query.
2. Step 2 is initialization that takes time linear in the length of random bits, which is $O(\log Q(m))$.

3. Step 3 is initialization that take time linear in the space used, which is $\tilde{O}(W_f^{1-\beta} + W_f^{\beta} \cdot \log Q(m))$.

4. Step 4 loops t times, each time invoking A_{in} and h_r once, which takes $t \cdot (O(\log T') + \tilde{O}(m)) = \tilde{O}(W_f)$ time in total.

5. Step 5a loops t times, each time invoking A_{DSR} $\tilde{O}(W_f^{1-\beta})$ times and h_r once, which takes $\tilde{O}(W_f^{1-\beta}) \cdot \text{polylog}T + \tilde{O}(m) = \tilde{O}(W_f^{1-\beta})$ time, plus multiple invocations of Step 5b that will be analyzed next.

6. For step 5b, we consider one iteration in step 5a over (k, i), which computes a block $b_{k+1,i}$. According to the block locality property (Theorem 4.23 Item 5), at most 5 blocks in B_k will be queried when running A_{DSR} on all the entries in a block $b_{k+1,i}$. Therefore, five registers $g_2, \ldots, g_6$ are sufficient to store all queried blocks, and each block is computed at most once. In other words, step 5c and h_r are called at most 5 times for each pair of loop variable (k, i).

7. For each call to step 5c, it takes $\tilde{O}(m)$ time to evaluate h_r and one oracle query to D.

8. Step 5 loops d' times, taking $d' \cdot t \cdot (\tilde{O}(W_f^{1-\beta}) + 5 \cdot (\tilde{O}(m) + \tilde{O}(m))) = \tilde{O}(d_f \cdot W_f)$ time in total.

9. Step 6 loops n times, each time invoking step 5b once. Note that calling step 5b in step 6 does not satisfy the block locality property. However, the sequential order of queries guarantees that it queries at most $O(n/m)$ blocks in B_{d+1}, and between two queries to the same block there is no query to any other block. That is, each block is computed at most once. Therefore, step 6 takes $O(n/m) \cdot \tilde{O}(m) = \tilde{O}(W_f^{1-\beta})$ time in total.

We conclude that Rec runs in time $T = \tilde{O}(d_f \cdot W_f)$.

Oracle Queries. For the number of oracle queries to D, we count step 1, step 4, step 5a and step 5c. Step 1 makes 1 query. Step 4 makes T^{β} queries, and step 5c is called at most 5 times for each pair of loop variable (k, i) in step 5a, which makes $5 \cdot d' \cdot t = O(d_f \cdot W_f^{\beta})$ queries in total. Step 5a makes one query in each of the $d' \cdot t$ iterations, summing up to $\tilde{O}(d_f \cdot W_f^{\beta})$ queries in total. Therefore, the total number of oracle queries to D is $\tilde{O}(d_f \cdot W_f^{\beta})$. ■

4.3.4 Quantified derandomization from uniform assumptions

Our goal now is to deduce overhead-free quantified derandomization from a uniform hardness assumption, using the construction in Proposition 4.26. We first establish

this standard application, proving overhead-free quantified derandomization of algorithms with one-sided error (Theorem 4.29), and then we use Lemma 4.16 to deduce quantified derandomization of algorithms with two-sided error.

Theorem 4.29 (quantified derandomization of prRTC from uniform hardness assumptions)**.** *Let $T, Q, C, W : \mathbb{N} \to \mathbb{N}$ such that $n \leq Q(n) \leq 2^{n^{1-\epsilon}}$ for a constant $\epsilon > 0$, and such that $C(n) = W(n)^{1-\beta}$ for a constant $\beta > 0$.*

Assume that there exists a function $f : \{0,1\}^n \to \{0,1\}^n$ that can be computed by a sufficiently uniform circuit family of width $W(n)$ and depth $d(n) = \text{polylog}(W(n))$, and is hard against $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}[T', S]$ on all but finitely many inputs, where

$$T'(n) = \tilde{O}(W + W^{\beta} \cdot T),$$
$$S(n) = \tilde{O}(W^{1-\min\{\beta, \epsilon\cdot(1-\beta)\}} + T).$$

Let $\Pi \in \mathsf{prRTC}_Q[T, C]$ be a promise problem that has a verifier $V(x; r)$ satisfying

$$\forall x : \min_{\sigma \in \{0,1\}} |\{r : V(x; r) = \sigma\}| \leq Q(C),$$

then

$$\Pi \in \mathsf{prDTIME}[\tilde{O}(T \cdot W^{\beta} + W)].$$

Proof. For a promise problem $\Pi \in \mathsf{prRTC}_Q[T, C]$, we let $A(x; r)$ be the procedure that takes input $x \in \{0,1\}^n$, random bits $r \in \{0,1\}^{C(n)}$ and decides Π for all but at most $Q(C(n))$ bad choices of r. We instantiate Proposition 4.26 with f and parameters ϵ, β, W, d. The obtained procedure H_f gets input x and outputs a list of C-bit strings with size $\tilde{O}(W^{\beta})$ in $\tilde{O}(W)$ time.

For every x we define a distinguisher D_x as $D_x(r) := A(x; r)$, which rejects at most $Q(C)$ strings if $x \in \Pi_Y$, and rejects all strings if $x \in \Pi_N$, and for x outside the promise has at most $Q(C)$ exceptional inputs. Suppose for the sake of contradiction that D_x rejects all strings that $H_f(x)$ prints for infinitely many $x \in \Pi_Y$. Then, for every such x, the procedure $\mathsf{Rec}^{D_x}(x)$ from Proposition 4.26 computes $f(x)$; and Rec^{D_x} is consistent with f on all inputs (using the fact that for each x the oracle D_x is in $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1$, as defined in Proposition 4.26).

In terms of complexity, Rec runs in time $\tilde{O}(d \cdot W)$ and makes $\tilde{O}(d \cdot W^{\beta})$ many queries to D_x, and thus its total running time is $\tilde{O}(W^{\beta} \cdot T + W)$. Moreover, Rec uses space $\tilde{O}(W^{1-\beta} + W^{1-\epsilon(1-\beta)})$, along with queries to D_x, where the $O(T)$ space it uses for computing D_x can be re-used across queries. Therefore, the total space is $\tilde{O}(W^{1-\min\{\beta, \epsilon\cdot(1-\beta)\}} + T)$. This contradicts the hardness assumption for f.

We conclude that for all but finitely many $x \in \Pi_Y$ at least one string in $H_f(x)$ is accepted by $D_x(\cdot) = A(x, \cdot)$, and for every $x \in \Pi_N$ all strings in $H_f(x)$ are rejected by D_x. We derandomize A by enumerating over outputs of H_f, which takes time $\tilde{O}(W)$ (to run $H_f(x)$) plus $\tilde{O}(W^{\beta} \cdot T)$ (to run A on each of the $\tilde{O}(W^{\beta})$ outputs of $H_f(x)$ and take an OR). ∎

Combining Theorem 4.29 with Lemma 4.16 and Theorem 4.2, we prove Theorem 1.6, which deduces overhead-free quantified derandomization for algorithms with two-sided error $\delta(T) = 2^{-\Omega(T)}$ from uniform assumptions.

Theorem 4.30 (Theorem 1.6, restated)**.** *For every increasing time-constructible function $T : \mathbb{N} \to \mathbb{N}$ such that $T(n) \geq n$ and sufficiently small $\epsilon > 0$, the following holds. Assume that for both $T_0(n) = T(n)$ and $T_0(n) = n$ there is $f_{T_0} : \{0,1\}^n \to \{0,1\}^n$ that can be computed by a sufficiently uniform circuit family of width $W_{T_0} = T_0(n)^{1+O(\epsilon)}$ and depth* $\mathrm{polylog}(T_0)$*, and is hard against the class*

$$\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}[W_{T_0}^{1+O(\epsilon)}, W_{T_0}^{1-\epsilon^4/2}]$$

on all but finitely many inputs. Then

$$\mathsf{BPTIME}_{2^{(1-\epsilon)T}}[T] \subseteq \mathsf{DTIME}[T^{1+O(\epsilon)}].$$

Proof. We first apply Theorem 4.2 with parameter ϵ, which gives us

$$\mathsf{BPTIME}_{2^{(1-\epsilon)T}}[T] \subseteq \mathsf{BPTC}_{2^{C^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C = T^{1+O(\epsilon^2)}].$$

By Lemma 4.16, every promise problem $\Pi \in \mathsf{BPTC}_{2^{C^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C]$ can be decided by an $\mathsf{RTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C/2]$ algorithm A with one $\mathsf{prcoRTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T_{\mathtt{pad}}, O(n)]$ oracle query to input length $N = n + C/2$, where $T_{\mathtt{pad}}(N) = N^{1+O(\epsilon)}$.

We first derandomize the prcoRTC problem in the oracle (to obtain a prRTC algorithm without an oracle query). Specifically, we denote the input length by $N = n + C(n)/2$, and let $C' = O(N)$ be the number of coins used by the prcoRTC algorithm. We instantiate Theorem 4.29 with parameter $\epsilon' = \epsilon^4$, error bound $Q(C) = 2^{(2C)^{1-O(\epsilon^3)}} \leq 2^{C^{1-\epsilon'}}$, time bound $T' = T_{\mathtt{pad}}$, and C' coins; and with our hypothesized hard function for $T_0(N) = N$, which is computable by circuits of width $W_1(N) = N^{1+O(\epsilon)} = (C')^{1/(1-\beta)}$, where $\beta = \Theta(\epsilon)$. The required hardness in Theorem 4.29 is against $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}$ with time $\tilde{O}(W_1^{\beta} \cdot T_{\mathtt{pad}}) = N^{1+O(\epsilon)}$ and space

$$\tilde{O}\left(W_1^{1-\min\{\beta, \epsilon' \cdot (1-\beta)\}} + T_{\mathtt{pad}}\right) \leq W_1^{1-\epsilon^4/2},$$

where $T_{\mathtt{pad}}$ is dominated in the expression above because $\epsilon' = \epsilon^4 \ll \beta, \epsilon$, and assuming that the constant hidden inside the O-notation in our hypothesized upper bound $W_1(N) = N^{1+O(\epsilon)}$ is larger than the universal constant hidden inside the O-notation in $T_{\mathtt{pad}}(N) = N^{1+O(\epsilon)}$ from Lemma 4.16. This hardness matches our assumption, so we conclude that we can derandomize the oracle problem in time $N^{1+O(\epsilon)} \leq T^{1+O(\epsilon)}$, where the inequality plugs in the input length $N = n + C/2 \leq T^{1+O(\epsilon^2)}$.

Now we need to derandomize $\mathsf{RTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C/2]$. We instantiate Theorem 4.29 with $\epsilon' = \epsilon^4$, error bound $Q(C) = 2^{(2C)^{1-O(\epsilon^3)}} \leq 2^{C^{1-\epsilon'}}$, time bound $T^{1+O(\epsilon)}$, and $C' = C/2$ coins; and with our hypothesized hard function for $T_0 = T$, which

is computable by circuits of width $W_2 = T^{1+O(\epsilon)} = (C')^{1/(1-\beta)}$, where $\beta = \Theta(\epsilon)$. The required hardness in Theorem 4.29 is against $\mathsf{F}(\mathsf{MA} \cap \mathsf{AM})\mathsf{TISP}$ with time $\tilde{O}(W_2^{\beta} \cdot T^{1+O(\epsilon)}) \leq T^{1+O(\epsilon)}$ and space

$$\tilde{O}\left(W_2^{1-\min\{\beta,\epsilon'\cdot(1-\beta)\}} + T^{1+O(\epsilon)}\right) \leq W_2^{1-\epsilon^4/2},$$

where again $T^{1+O(\epsilon)}$ is dominated if we choose the constant hidden inside the O-notation of the upper bound W_2 to be large enough.[33] This hardness matches our assumption, so we conclude that

$$\mathsf{RTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C/2] \subseteq \mathsf{DTIME}[T^{1+O(\epsilon)}].$$

Combining all of the above, we deduce that

$$\begin{aligned}
\mathsf{BPTIME}_{2^{(1-\epsilon)T}}[T] &\subseteq \mathsf{BPTC}_{2^{C^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C = T^{1+O(\epsilon^2)}] && \text{(Theorem 4.2)}\\
&\subseteq \mathsf{RTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C/2]^{\mathsf{prcoRTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T_{\mathsf{pad}}, O(n)]}\\
&\qquad \text{(one oracle query to input length } n + C/2\text{; Lemma 4.16)}\\
&\subseteq \mathsf{RTC}_{2^{(2C)^{1-O(\epsilon^3)}}}[T^{1+O(\epsilon)}, C/2] && \text{(Theorem 4.29)}\\
&\subseteq \mathsf{DTIME}[T^{1+O(\epsilon)}], && \text{(Theorem 4.29 again)}
\end{aligned}$$

as we wanted. ■

5 Targeted Samplers Based on Non-batch-computable Functions

In the following statement we assume a function $\{0,1\}^n \times \{0,1\}^r \to \{0,1\}^k$ that is non-batch-computable for algorithms running in time $T' \approx T \cdot k$. That is, each output bit is computable in time T', but to compute the entire k-bit output, one must invest time at least $k^{\Omega(1)} \cdot T'$. The crucial point is that the hardness assumption holds for every $x_1 \in \{0,1\}^n$ and at least $1 - \delta$ of $x_2 \in \{0,1\}^r$.

Theorem 5.1 (targeted sampler based on non-batch computability)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $T, r, k : \mathbb{N} \to \mathbb{N}$ be polynomials such that $r(n) \leq T(n) \cdot k(n)$. Assume that one-way functions exists, and assume that for $\delta : \mathbb{N} \to (0,1)$ there exists a two-input function $f : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{k(n)}$ satisfying the following.*

- ***Upper bound.*** *There is a deterministic algorithm that for all $x_1 \in \{0,1\}^n$ and for a $(1-\delta)$ fraction of $x_2 \in \{0,1\}^r$, when given input (x_1, x_2, i) the algorithm runs in time $T'(n) = T(n) \cdot k(n)$ and outputs $f(x_1 \circ x_2)_i$.*

[33] This constant now needs to be large enough to compensate for the overheads feeding into the time bound $T^{1+O(\epsilon)}$ obtained after the derandomization of the oracle problem; these overheads come both from Lemma 4.16 and from the time bound on the hypothesized hard function with $T_0(N) = N$.

- ***Non-batch computability.*** *There are small enough constants $\alpha, \beta \in (0,1)$ such that any randomized algorithm A running in time $T'(n) \cdot k^{\beta}$ satisfies the following: for all but finitely many x_1, for all but δ fraction of $x_2 \in \{0,1\}^r$, with probability at least $1/2$ over internal randomness of A, the output of $A(x_1 \circ x_2)$ disagrees with $f(x_1 \circ x_2)$ on at least α fraction of indices $i \in [k]$.*

Then, there exists a $(k^{-\Omega_{\alpha,\beta}(1)}, 2\delta)$-targeted sampler against T-time algorithms with sample complexity $S \leq k^c$ and randomness complexity r such that the total time for printing all the samples is at most $T \cdot k^c$.

Proof. We are going to use the reconstructive generator of [NW94; IW97] as a learning algorithm presented in [CT21a].

Theorem 5.2 (The reconstructive generator of [NW94; IW97] as a learning algorithm; see, e.g., Theorem 6.3 in [CT21a])**.** *For every constants $\alpha_0, \beta_0 \in (0,1)$, there exists an oracle machine Gen_0 and a probabilistic machine Rec_0 such that for any function $g : \{0,1\}^{\log k} \to \{0,1\}$ and sufficiently small $\gamma_0 = \gamma_{\alpha_0,\beta_0} > 0$ the following holds.*

1. ***Generator.*** *Given oracle access to g, the generator $\mathsf{Gen}_0^g(1^k, \gamma_0)$ runs in time $\mathrm{poly}(k)$ and outputs a list L of m_0-bit strings, where $m_0 = k^{\gamma_0}$.*
2. ***Reconstruction.*** *Given oracle access to g and a $(1/m_0)$-distinguisher D_0 for L, the algorithm $\mathsf{Rec}_0^{g,D_0}(1^k, \gamma_0)$ runs in time $O(k^{\beta_0})$ and with probability at least $1 - 2^{-2m_0}$ it outputs an oracle circuit C^{D_0} such that $\Pr_{z \in \{0,1\}^k}[C^{D_0}(z) = g(z)] \geq 1 - \alpha_0$.*

Let c be a large enough constant whose value will be determined as we go through the proof. Let α, β be the sufficiently small constants given in the non-batch computability assumption. Then, in Theorem 5.2, we may choose constants $\alpha_0, \beta_0, \gamma_0$ small enough such that $\alpha_0 < \alpha$ and $2(\beta_0 + \gamma_0) < \beta$. Let Gen_0 and Rec_0 be the corresponding generator and reconstruction pair. Also, let $\mu > 0$ be a small enough constant such that $T^{\mu} = k^{\gamma_0}$. Assuming the existence of one-way functions, it follows from Theorem 3.3 that there exists a cryptographic PRG $G_{\text{crypt}} : \{0,1\}^{T^{\mu}=k^{\gamma_0}} \to \{0,1\}^T$ that runs in time $T^{1+\mu} = T \cdot k^{\gamma_0}$ and is secure against polynomial-size circuits.

Sampler. Given input $x \in \{0,1\}^n$ and random string $y \in \{0,1\}^r$, we define our sampler $\mathsf{Samp}(x,y)$ as follows: it first generates the list of strings $L^{x \circ y} := \mathsf{Gen}_0^{f(x \circ y)}(1^k, \gamma_0)$, and then applies G_{crypt} to each of the strings in $L^{x \circ y}$; i.e.,

$$\mathsf{Samp}(x,y) := \{G_{\text{crypt}}(z) : z \in L^{x \circ y}\}.$$

Note that Samp has randomness complexity $|y| = r$ and output length T. Additionally, the number of strings $|L^{x \circ y}|$ is at most the running time of Gen_0, which is $S \stackrel{\text{def}}{=\!=} \mathrm{poly}(k) \leq k^c$ for a large enough constant c; therefore, the sample complexity (the number of outputted strings) of $\mathsf{Samp}(x,y)$ is at most $S \leq k^c$ as well.

As for the running time, computing $f(x \circ y)$ (bit by bit) takes time $O(T \cdot k^2)$, and Gen_0 runs in time $\mathrm{poly}(k)$, and applying G_{crypt} on all the S strings takes time $(T \cdot k^{\gamma_0}) \cdot \mathrm{poly}(k)$. Therefore, printing all outputs of Samp can be done in time $T \cdot \mathrm{poly}(k)$.

Correctness. Next we prove that Samp is a $(2k^{-\gamma_0}, 2\delta)$-targeted sampler. For the sake of contradiction, suppose not. Then, there exists a T-time targeted distinguisher $D : \{0,1\}^n \times \{0,1\}^T \to \{0,1\}$ such that for infinitely many inputs $x \in \{0,1\}^n$

$$\Pr_{y\sim\{0,1\}^r}\left[\left|\mathbb{E}_{z\sim\mathsf{Samp}(x,y)}[D(x,z)] - \mathbb{E}_{z\sim\mathcal{U}_T}[D(x,z)]\right| > 2k^{-\gamma_0}\right] > 2\delta. \tag{2}$$

We organize the rest of the proof as follows: First, we use D to construct a distinguisher D_0 for Gen_0 as in the following claim; we then use D_0 combined with the reconstruction algorithm Rec_0 to construct a randomized algorithm B that contradicts the non-batch-computability assumption.

Claim 5.3. *For any fixed $x \in \{0,1\}^n$, let $D_x\colon \{0,1\}^{k^{\gamma_0}} \to \{0,1\}$ be defined by $D_x(z) = D(x, G_{crypt}(z))$. Then, for infinitely many inputs x, and at least 2δ fraction of $y \in \{0,1\}^{r(|x|)}$, D_x is an $(k^{-\gamma_0})$-distinguisher for $L^{x\circ y} = \mathsf{Gen}_0^{f(x\circ y)}(1^k, \gamma_0)$.*

Proof. For any x satisfying Eq. (2), let $Y_x \subseteq \{0,1\}^r$ with $|Y_x| \geq 2\delta \cdot 2^r$ denote the set of strings $y \in \{0,1\}^r$ such that $\left|\mathbb{E}_{z\sim L^{x\circ y}}[D(x, G_{\text{crypt}}(z)] - \mathbb{E}_{z\sim\mathcal{U}_T}[D(x,z)]\right| > 2k^{-\gamma_0}$. Then, for any $y \in Y_x$ we have

$$\begin{aligned}
&\mathbb{E}_{z\sim L^{x\circ y}}[D_x(z)] - \mathbb{E}_{z\sim\mathcal{U}_{k^{\gamma_0}}}[D_x(z)] \\
&\qquad \geq \left|\mathbb{E}_{z\sim L^{x\circ y}}[D(x, G_{\text{crypt}}(z))] - \mathbb{E}_{z\sim\mathcal{U}_T}[D(x,z)]\right| \\
&\qquad\quad - \left|\mathbb{E}_{z\sim\mathcal{U}_T}[D(x,z)] - \mathbb{E}_{z\sim\mathcal{U}_{k^{\gamma_0}}}[D(x, G_{\text{crypt}}(z))]\right| \\
&\qquad > 2k^{-\gamma_0} - \text{negl}(n) \\
&\qquad \geq k^{-\gamma_0}
\end{aligned}$$

where the penultimate inequality is since $y \in Y_x$ and due to the security of G_{crypt} against the polysize circuit $D'_x(z) = D(x,z)$, and the last inequality is sinde $k = \text{poly}(n)$. ■

Next, for inputs $x \in \{0,1\}^n$ and $y \in \{0,1\}^{r(|x|)}$, we define $A(x \circ y)$ to work as follows:

- First run the reconstruction algorithm $\mathsf{Rec}_0^{D_x(\cdot), F(x\circ y,\cdot)}(1^k, \gamma_0)$ to obtain an oracle circuit C^{D_x}, where D_x is defined in Claim 5.3, and $F(x \circ y, i)$ is the algorithm computing $(x, y, i) \mapsto f(x \circ y)_i$ from the upper bound assumption.
- Next, for every $i \in [k]$ print the ith output bit by computing $C^{D_x}(i)$.

Batch computability. By Claim 5.3, for infinitely many inputs x, and at least 2δ fraction of $y \in \{0,1\}^{r(|x|)}$, D_x is a $k^{-\gamma_0}$-distinguisher for $\mathsf{Gen}_0^{f(x\circ y)}(1^k, \gamma_0)$. By the upper-bound assumption, for at least $(1-\delta)$ fraction of $y \in \{0,1\}^{r(|x|)}$, the algorithm $F(x \circ y, i)$ correctly computes $f(x \circ y, i)$ for all i.

Hence, for at least δ fraction of strings y both of the above hold. For each such y, by Theorem 5.2, with probability at least $(1 - 2^{-2k^{\gamma_0}}) > 1/2$ over the randomness of Rec_0, the outputted circuit $C^{D_x}(i)$ computes $f(x \circ y)_i$ for at least $(1-\alpha)$ fraction of $i \in [k]$.

Time complexity. It only remains to bound the time complexity of A. Recall that the oracle algorithm Rec_0 runs in time $O(k^{\beta_0})$, and therefore, the oracle circuit C also has size at most $O(k^{\beta_0})$. In addition, resolving each query to $F(x \circ y, i)$ takes time $T \cdot k$, and each query to $D_x(z) = D(x, G_{\text{crypt}}(z))$ is computable in time $T_0 = O(T \cdot k^{\gamma_0})$.

As a result, simulating $\mathsf{Rec}_0^{D_x,F}$ can be done in time

$$O(k^{\beta_0})(T \cdot k^{\gamma_0} + T \cdot k) = O(T \cdot k^{1+\beta_0+\gamma_0}),$$

and running $C^{D_x}(i)$ for $i \in [k]$ takes time

$$k \cdot O(k^{\beta_0}) \cdot (T \cdot k^{\gamma_0}) = O(T \cdot k^{1+\beta_0+\gamma_0})$$

as well. Therefore, $A(x \circ y)$ is computable in total time $O(T \cdot k^{1+\beta_0+\gamma_0}) < T \cdot k^{1+\beta/2}$, contradicting the non-batch computability assumption. ∎

In particular, instantiating Theorem 5.1 with parameter values $k = n^{\epsilon}$ and $\alpha = \beta = 0.01$, we obtain Theorem 1.7 as a corollary.

Corollary 5.4 (targeted samplers with small output-set from non-batch-computability; Theorem 1.7, restated formally)**.** *Let $T(n) = \text{poly}(n)$, let $\epsilon \in (0,1)$ be a constant, and let $\delta \colon \mathbb{N} \to (0,1)$. Assume that OWFs exist, and that there is $f : \{0,1\}^n \times \{0,1\}^{O(T)} \to \{0,1\}^{n^{\epsilon}}$ satisfying the following.*

- ***Upper bound.*** *There is a deterministic algorithm that computes the mapping $(x_1, x_2, i) \mapsto f(x_1, x_2)_i$ in time $T' = T \cdot n^{\epsilon}$.*
- ***Non-batch computability.*** *Any randomized algorithm A running in time $T' \cdot n^{\epsilon/100}$ satisfies the following: for all but finitely many x_1, for all but a δ fraction of $x_2 \in \{0,1\}^{r(|x_1|))}$, with probability at least $1/2$ over internal randomness of A, the output of $A(x_1 \circ x_2)$ disagrees with $f(x_1 \circ x_2)$ on at least a .01 fraction of indices.*

Then, there is an $(o(1), \delta)$-targeted sampler against T-time algorithms that has sample complexity $n^{O(\epsilon)}$ and randomness complexity $O(T)$ such that the total time for printing all the samples is at most $T \cdot n^{O(\epsilon)}$.

Note that Theorem 5.1 yields a targeted sampler with error 2δ. However, inspecting the proof, when the upper-bound holds for all (x_1, x_2) rather than only for $1 - \delta$ of x_2's, the final error of the sampler is δ as stated in Corollary 5.4, rather than 2δ.

6 Targeted Hitters Based on Circuits with Low-degree Gates

In this section we construct targeted hitters based on a hardness assumptions for functions computable by uniform circuits with low-degree gates.

We first fix some notation and present the definition of uniform circuits with low-degree gates. Then, in Section 6.1 we present a construction of a reconstructive generator based on hard functions computable by uniform circuits with low-degree gates, and in Section 6.2 we present the construction of a targeted hitter based on the foregoing reconstructive generator.

Circuits with arithmetic gates. We will consider uniform circuits whose gates compute arithmetic circuits $\mathbb{F}_2^m \to \mathbb{F}_2$ of bounded degree, defined as follows:

In the following definition, the main asymptotic parameter is denoted by $N \in \mathbb{N}$. (When using the definition later, the parameter $N = N(n)$ will be a function of a hidden asymptotic parameter $n \in \mathbb{N}$.) The parameters w and d are the width and depth of the circuit, respectively. For generality we also account for the fan-in of gates above the bottom layer, although in our final result we will not limit their fan-in.

Definition 6.1 (uniform circuit with low-degree gates)**.** *For $w, d, k : \mathbb{N} \to \mathbb{N}$, we say that $\{C_N : \{0,1\}^N \to \{0,1\}^w\}$ is computable by a* highly uniform family of $(d \times w)$-circuits with (Δ, T)-low-degree universal gates of fan-in *k if:*

- **(Circuit structure.)** *The circuit has d layers, and in each layer there are w gates, each taking inputs from the previous layer. There is an algorithm that gets input 1^N, runs in time $\tilde{O}(T)$, and prints the description of an arithmetic circuit G over $\mathbb{F}_2$ of size T and with degree Δ that the gates compute.*[34] *Gates in the first layer have unbounded fan-in, and gates in subsequent layers have fan-in at most k.*
- **(Uniformity.)** *There is a $\mathcal{P}$-uniform family of depth-$O(\log\log(w))$ formulas that, given $(i, g, j) \in [d] \times [w] \times [N]$, outputs the the index of the jth child of gate g in layer i.*

An useful parameter setting to keep in mind, which will be the one used in our results, is $d = w \approx N^{\epsilon}$ and $\Delta \approx N^{\epsilon^2}$ (and an unlimited fan-in k).

Remark 6.2. *At first, requiring that all gates compute the same circuit G may seem overly restrictive, and it might be tempting to allow the g-th gate to compute $F(g, \cdot)$ for some low-degree arithmetic circuit. However, this additional flexibility can easily be simulated using a single universal gate: adding two axillary gates to each layer that constantly output 0 and 1, respectively, we can wire a set of $\log w$ additional wires for the g-th gate to the 0 and 1 gates so that they encode the binary representation of g.*

6.1 A reconstructive targeted generator

In the following, we provide our new reconstructive generator.

Theorem 6.3 (reconstructive generator based on circuits with low-degree gates)**.** *Let $T, w, d, k, \Delta : \mathbb{N} \to \mathbb{N}$ be computable in linear time such that $w(N)$ is always a power of 2 and $w(N) \cdot T(N) \geq N$,*[35] *and let $\epsilon \in (0,1)$ be a small enough constant. Let $\{C_N : \{0,1\}^N \to$*

[34] Indeed, we assume that all gates compute the same arithmetic circuit, and only the wiring changes; this assumption loses less generality than one might expect (see Remark 6.2). We need to account for the fact that different gates may have different fan-ins, and our proof carries through with any reasonable way of handling this. For concreteness, we assume that the general version of G takes N inputs, and the wiring function can determine that some inputs are fixed to constants; as said, this choice does not matter for our proof, and it can be replaced by (say) having a different arithmetic circuit for each layer, with a different fan-in per layer (as long as the circuit for each layer can be printed in time $\tilde{O}(T)$).

[35] Strictly speaking, the condition $w(N) \cdot T(N) \geq N$ is not necessary. It is included simply to assist with the parsing of the parameters by ensuring that the first layer of the circuit has enough time to read all the input bits.

$\{0,1\}^w\}$ *be computable by a highly uniform family of* $(d \times w)$*-circuits with* (Δ, T)*-low-degree gates of fan-in* k*. Then, there are algorithms* Gen, Rec *satisfying the following.*

- **(Somewhere generator.)** *Given input* $x \in \{0,1\}^N$*, the generator* $\mathsf{Gen}(x)$ *runs in time* $d \cdot \Delta^{3/\epsilon} \cdot w^{2+O(\epsilon)} \cdot \max\{\tilde{O}(T), w\}$ *and outputs* $d' = O(d/\epsilon)$ *lists of* M*-bit strings* $L^{(x)} := \bigcup_{i \in [d']} L_i^{(x)}$ *where* $M = (\Delta^{1/\epsilon} \cdot w)^{\Omega(\epsilon^2)}$ *and* $|L^{(x)}| \leq d \cdot \Delta^{3/\epsilon} \cdot w^{2+O(\epsilon)}$*.*

- **(Reconstruction.)** *Suppose that* Rec *receives input* $x \in \{0,1\}^N$ *and oracle access to a function* $D \colon \{0,1\}^M \to \{0,1\}$ *such that for every* $i \in [d']$ *we have*

$$|\mathbb{E}_{y \sim \{0,1\}^M}[D(y)] - \mathbb{E}_{y \sim L_i^{(x)}}[D(y)]| \geq 1/M\,.$$

  Then, $\mathsf{Rec}^D(x)$ *runs in time* $(w+d) \cdot \mathrm{poly}(\Delta, w^\epsilon) \cdot \max\{k, \tilde{O}(T)\}$*, makes at most* $(\Delta \cdot w^\epsilon)^c \cdot (w + d \cdot k)$ *queries to* D*, and with probability at least* 0.9 *outputs* $C_N(x)$*, where* $c > 1$ *is a universal constant.*

Our proof proceeds in two steps.

1. In Section 6.1.1 we show an efficient decomposition of the layers of the computation of $C_N(x)$ into a list of low-degree polynomials, where the first polynomial is locally computable from the input x, and the remaining polynomials are downward self-reducible.

2. In Section 6.1.2 we construct the generator and the reconstruction algorithm. The generator will apply a (variant of) a classical pseudorandom generator to each of the polynomials in the foregoing list, and the reconstruction will use the learning algorithm and the downward self-reducibility of the polynomials.

For context, this high-level plan follows the high-level plan in [CT21a], but the technical details are very different: we are working with a more restricted type of circuits (i.e., circuits with low-degree gates), but our generator and reconstruction run in time that is near-linear in T (when the other parameters d, w, k, Δ are small).

6.1.1 Polynomial decomposition of uniform circuits with low-degree gates

In this section, we provide a polynomial decomposition for our notion of uniform circuits with low degree gates, which is analogous in spirit to the bootstrapping systems of [CT21a] and the polynomial decomposition of [CT21a].

Proposition 6.4 (polynomial decomposition of uniform circuits with low degree gates)**.** *In the following:*

- *Let* $w, d, \Delta, T, k : \mathbb{N} \to \mathbb{N}$ *be time-constructible* $w(N)$ *is a power of* 2*.*

- *Let* $\{C_N : \{0,1\}^N \to \{0,1\}^w\}$ *be computable by a highly uniform family of* $(d \times w)$*-circuits with* (Δ, T)*-low-degree gates.*

- *Let $h, m \colon \mathbb{N} \to \mathbb{N}$ such that $h \geq \log(w)$ and $h^m = w$, and let $\{\mathbb{F}_N\}$ be any family of finite fields with characteristic 2 where $h \leq |\mathbb{F}_N| \leq \mathrm{poly}(w)$.*

Then, for any $x \in \{0,1\}^N$ there exists a list of $d' = O(d \cdot m)$ polynomials $\{P_i^{(x)} : \mathbb{F}_N^m \to \mathbb{F}_N\}_{i \in [d']}$ such that the following holds.

(i) ***Degree.*** *Each polynomial has total degree at most $\Delta' = \Delta \cdot h^2 \cdot \mathrm{polylog}(w)$.*

(ii) ***First layer.*** *There is an algorithm* Base *that given inputs $x \in \{0,1\}^N$ and $v \in \mathbb{F}_N^m$ outputs $P_1^{(x)}(v)$ in time $w \cdot (\tilde{O}(T) + \mathrm{poly}(h))$.*

(iii) ***DSR.*** *There is an oracle machine* DSR *that takes inputs $i \in [d' - 1]$ and $v \in \mathbb{F}_N^m$, makes $\max\{k, h\}$ queries to $P_i^{(x)}$, runs in time $\tilde{O}(T) + k \cdot \mathrm{poly}(h)$, and outputs $P_{i+1}^{(x)}(v)$.*

(iv) ***Final layer.*** *There is an oracle machine* Final *that takes an input $j \in [w]$, makes an oracle query to $P_{d'}^{(x)}$, and outputs $C_N(x)_j$ in time $\mathrm{poly}(h)$.*

(v) ***Print.*** *There is an algorithm* Print *that given $x \in \{0,1\}^N$, outputs the truth-tables of all polynomials, i.e. $\{P_i^{(x)}(v)\}_{v \in \mathbb{F}_N^m, i \in [d']}$, in time $d \cdot |\mathbb{F}_N|^m \cdot \mathrm{poly}(h) \cdot \max\{w, \tilde{O}(T)\}$.*

The proof is structured as follows. After defining the arithmetic setting, we show the construction of the first polynomial, then a lifting procedure for going from layer i to $i + 1$, and then a degree-reduction procedure for resetting the degree. We will finally combine these procedures to complete the proof.

For the rest of this proof, for a fixed input $x \in \{0,1\}^N$ and $i \in [d]$, we use $\alpha_i^{(x)} : \{0,1\}^{\log w} \to \{0,1\}$ to denote the value of the gates in layer i of $C_N(x)$. We may drop the superscript x whenever it is obvious from the context.

Arithmetic setting. Let $H \subset \mathbb{F}$ be an $\mathbb{F}_2$-subspace of $\mathbb{F} = \mathbb{F}_N$ with size $|H| = h$; specifically, $H = \{\eta(z) : z \in \{0,1\}^{\log h}\}$ where $\eta \colon \{0,1\}^{\log h} \to H$ is defined as $\eta(z) := \sum_{i=1}^{\log h} z_i \cdot e_i$ such that e_i is the i-th standard element of $(\mathbb{F}_2)^{\log |\mathbb{F}|}$.

We can extend the bijection η to $\eta^{\otimes m} : \{0,1\}^w \longleftrightarrow H^m$ where for $z \in \{0,1\}^w$ we first partition $z = z^{(1)} \circ \cdots \circ z^{(m)} \in \{0,1\}^{m \cdot \log h}$ and then take the direct product $\eta^{\otimes m}(z) := (\eta(z^{(1)}), \ldots, \eta(z^{(m)})) \in H^m$. We note that $\eta^{\otimes m}(\cdot)$ is computable in time $\mathrm{poly}(m, \log |\mathbb{F}|) = \mathrm{poly}(h)$. With a slight abuse of notation, we use η to denote $\eta^{\otimes m}$ whenever the meaning is clear from the context.

For $u \in H$, the Kronecker delta function $\delta_u : \mathbb{F} \to \mathbb{F}$ is defined as $\delta_u(v) := \prod_{u' \in H \setminus u} \frac{v - u'}{u - u'}$. Recall that δ_u has maximum individual degree $h - 1$ over $\mathbb{F}$, satisfies

$$\delta_u(v) = \begin{cases} 1 & v = u, \\ 0 & v \neq u \end{cases}$$

for $v \in H$, and is computable in time $\mathrm{poly}(h, \log(|\mathbb{F}|)) = \mathrm{poly}(h)$.

First polynomial. We construct a low-degree polynomial P_1 that agrees with the first layer $\alpha_1^{(x)}$ of the circuit on $H^m \equiv \{0,1\}^{\log w}$.

Lemma 6.5 (first polynomial)**.** *Given input $x \in \{0,1\}^N$, there exists a polynomial $P_1^{(x)} : \mathbb{F}^m \to \mathbb{F}$ that satisfies the following.*

(a) ***Agreement.*** *For $z \in \{0,1\}^{\log w}$ we have $P_1^{(x)}(\eta(z)) = \alpha_1^{(x)}(z)$.*

(b) ***Degree bound.*** *The maximum individual degree of $P_1^{(x)}$ is at most $h-1$.*

(c) ***Local computability.*** *There exists an algorithm that given input $x \in \{0,1\}^N$ performs a preprocessing in time $w \cdot \tilde{O}(T)$, and then, for any $v \in \mathbb{F}^m$ outputs $P_1^{(x)}(v)$ in time $w \cdot \mathrm{poly}(h)$. In particular, $P_1^{(x)}$ is computable in time $w \cdot (\tilde{O}(T) + \mathrm{poly}(h))$.*

Proof. Let $P_1 : \mathbb{F}^m \to \mathbb{F}$ be the low-degree extension of $\alpha_1^{(x)}$ defined as

$$P_1(v) := \sum_{u \in H^m} \alpha_1^{(x)}(\eta^{-1}(u)) \cdot \prod_{a \in [m]} \delta_{u_a}(v_a). \tag{3}$$

By definition of δ_{u_a}, this polynomial has maximum individual degree at most $h-1$ and for $z \in \{0,1\}^{\log w}$ evaluates to $P_1(\eta(z)) = \alpha_1^{(x)}(z)$.

Given $x \in \{0,1\}^N$ and $v \in \mathbb{F}^m$, our algorithm will compute the truth-table of $\alpha_1^{(x)}$, and then calculate Equation (3). Specifically, for $z \in \{0,1\}^{\log w}$, the value of $\alpha_1^{(x)}(z)$ is computable in time $\tilde{O}(T)$ from x,[36] and hence the truth-table of $\alpha_1^{(x)}$ can be computed in time $w \cdot \tilde{O}(T)$. Then, given $v \in \mathbb{F}^m$, we compute $P_1(v)$ as in Equation (3) in time

$$\tilde{O}(w) \cdot m \cdot \mathrm{poly}(h) \leq w \cdot \mathrm{poly}(h) \,. \quad \blacksquare$$

Lifting to the next layer. Next, given a polynomial of maximum individual degree at most $h-1$ for the i-th layer, we show how to construct a polynomial of total degree $\Delta' = \Delta \cdot h \cdot \mathrm{polylog}(w)$ for the $(i+1)$-th layer that is DSR to the i-th layer. Of course, to be able to recursively apply this procedure, we have to deal with the degree blow-up, which will be done afterwards.

Lemma 6.6 (lifting from i to $i+1$)**.** *For $i \in \{1, \ldots, d-1\}$, Let $P_i^{(x)} : \mathbb{F}^m \to \mathbb{F}$ be a polynomial that agrees with $\alpha_i^{(x)}$ on $H^m \equiv \{0,1\}^{\log w}$ and has maximum individual degree at most $h-1$. Then, there exists a polynomial $P_{i+1}^{(x)} : \mathbb{F}^m \to \mathbb{F}$ that satisfies the following.*

(a) ***Agreement.*** *For $z \in \{0,1\}^{\log w}$ we have $P_{i+1}^{(x)}(\eta(z)) = \alpha_{i+1}^{(x)}(z)$.*

[36] Specifically, we use the uniformity condition to compute which indices of x feed into the gate represented by z. This takes time at most $T \cdot \mathrm{polylog}(T)$ as the gate has at most fan-in T. Then, we can compute the value of the gate itself in time $\tilde{O}(T)$, by evaluating the corresponding arithmetic circuit over $\mathbb{F}_2$.

(b) ***Degree.*** *The total degree of $P_{i+1}^{(x)}$ is at most $\Delta' = h^2 \cdot \Delta \cdot \mathrm{polylog}(w)$.*

(c) ***DSR.*** *There is an oracle machine that takes input $i \in \{2,...,d\}$ and $v \in \mathbb{F}^m$, makes k queries to $P_i^{(x)}$, runs in time $\tilde{O}(T) + k \cdot \mathrm{poly}(h)$, and outputs $P_{i+1}^{(x)}(v)$.*

Proof. The goal is to arithmetize the following computation of gate z in the $(i+1)$-th layer, which computes the function $G\colon \{0,1\}^k \to \{0,1\}$ as follows:

$$\alpha_{i+1}^{(x)}(z) = G\left(\alpha_i^{(x)}(\mathrm{IN}_{i,1}(z)), \ldots, \alpha_i^{(x)}(\mathrm{IN}_{i,k}(z))\right) \qquad (z \in \{0,1\}^{\log w}).$$

where $\mathrm{IN}_{i,j}(z)$ is the jth child of gate z in layer i.

By the uniformity assumption, the function mapping (i,z,j) to the jth child of z at layer i is computable by a $\mathcal{P}$-uniform formula of depth $O(\mathrm{loglog} w)$. Hence, we can print in time $\mathrm{polylog}(w)$ an *arithmetic* formula $\phi_{i,\ell} : \mathbb{F}_2^{\log w} \to \mathbb{F}_2^{\log w}$ of degree at most $\mathrm{polylog}(w)$ computing $z \mapsto \mathrm{IN}_{i,z,j}(z)$.[37] The following fact allows us to efficiently extend this formula to a low-degree polynomial $\hat{\phi}_{i,\ell}\colon \mathbb{F}^m \to \mathbb{F}^m$.

Claim 6.6.1. *Let $\phi : \{0,1\}^{m\cdot\log h} \to \{0,1\}^{m\cdot\log h}$ be computable in time τ such that ϕ has degree σ as a polynomial $\mathbb{F}_2^{m\cdot\log(h)} \to \mathbb{F}_2^{m\cdot\log(h)}$. Then, there exists a polynomial $\hat{\phi} : \mathbb{F}^m \to \mathbb{F}^m$ that is computable in time $\tau \cdot \mathrm{poly}(h)$, has maximum individual degree at most $\sigma \cdot h$, and for $z \in \{0,1\}^{m\cdot\log h}$ satisfies $\hat{\phi}(\eta(z)) = \eta(\phi(z))$.*

Proof. Recall that $\eta^{-1}\colon H \to \{0,1\}^{\log h}$ projects each H-element to its binary representation. We extend η^{-1} to a polynomial $\widehat{\eta^{-1}}\colon \mathbb{F} \to \mathbb{F}^{\log h}$ by the standard low-degree extension, which is of individual degree at most $h-1$. Let $\pi\colon \mathbb{F}^m \to \mathbb{F}^{m\cdot\log h}$ such that

$$\pi(v_1,...,v_m) = \widehat{\eta^{-1}}(v_1),...,\widehat{\eta^{-1}}(v_m)\ ,$$

and note that π is computable in time $m \cdot \mathrm{poly}(h) = \mathrm{poly}(h)$ (since $\widehat{\eta^{-1}}$ is computable in such time) and has maximum individual degree $h-1$.

We define $\hat{\phi}(v_1,...,v_m) := \eta(\phi(\pi(v_1,...,v_m)))$, wherein ϕ is evaluated as a polynomial $\mathbb{F}^{m\cdot\log h} \to \mathbb{F}^{m\cdot\log h}$. The maximum individual degree of $\hat{\phi}$ is at most $\sigma\cdot(h-1)$ (since η is linear and π has maximum individual degree $h-1$) and it is computable in time $\mathrm{poly}(h)\cdot\tau$. Finally, for $z \in \{0,1\}^{m\cdot\log h}$ we have $\pi(\eta(z)) = z$, hence $\hat{\phi}(\eta(z)) = \eta(\phi(\pi(\eta(z)))) = \eta(\phi(z))$. ■

Now, recall that $G\colon \mathbb{F}_2^k \to \mathbb{F}_2$ is a polynomial of degree Δ, and hence it is also a polynomial $\mathbb{F}^k \to \mathbb{F}$ of degree Δ. We define

$$P_{i+1}(v) := G\Big(\left(P_i(\hat{\phi}_{i,1}(v))\right),...,\left(P_i(\hat{\phi}_{i,k}(v))\right)\Big)\ .$$

[37] Specifically, the structure of the arithmetic formula mimics the structure of the Boolean formula (by replacing Boolean gates with arithmetic operations over $\mathbb{F}_2$), and this preserves $\mathcal{P}$-uniformity.

Note that P_{i+1} has total degree at most

$$\Delta \cdot (m \cdot (h-1)) \cdot (m \cdot h \cdot \text{polylog}(w)) \leq \Delta' \,.$$

The fact that P_{i+1} agrees with $\alpha_{i+1}^{(x)}$ on H^m follows since $\hat{\phi}_{i,\ell}$ agrees with $\text{IN}_{i,\ell}$ on H^m, and the output of $\hat{\phi}_{i,\ell}$ on H^m is in H^m, and P_i agrees with $\alpha_i^{(x)}$ on H^m.

It only remains to prove DSR. Given i and $v \in \mathbb{F}^m$ and oracle access to $P_i^{(x)}$, the algorithm computes the list of values $v_\ell := \hat{\phi}_{i,\ell}(v)$ for $\ell \in [k]$, which can be done in time $k \cdot \text{poly}(h)$. It then makes k queries to $P_i^{(x)}$, to obtain $\hat{b}_\ell := P_i^{(x)}(v_\ell) \in \mathbb{F}$. Finally, the algorithm computes $G(\hat{b}_1, ..., \hat{b}_k)$, by invoking the algorithm that prints G and simulating the circuit, which can be done in time $\tilde{O}(T) + T \cdot \text{polylog}(|\mathbb{F}|) = \tilde{O}(T)$. So, the total number of oracle queries is k, and the overall running time is at most $\tilde{O}(T) + k \cdot \text{poly}(h)$. ■

Degree reduction. Next, we need to reset the degree to $(h-1)$ before lifting the next polynomial. We will do that by adding "H-ization" polynomials (i.e., a-la linearization in [She92]) to reduce the individual degree of variables one-by-one, while preserving DSR and agreement on H^m.

Lemma 6.7 (degree reduction polynomials)**.** *For any polynomial $P : \mathbb{F}^m \to \mathbb{F}$ of total degree at most Δ' there exists a list of polynomials $Q_0 = P, Q_1, \ldots, Q_m$ satisfying the following:*

1. ***Agreement.*** *Q_m agrees with Q on H^m.*
2. ***Degree.*** *For $a \in \{0, \ldots, m-1\}$, the total degree of Q_a is at most Δ', and the maximum individual degree of Q_m is $h-1$.*
3. ***DSR.*** *There exists an algorithm that takes index $a \in [m]$ and $v \in \mathbb{F}^m$ as inputs, makes at most h queries to Q_{a-1}, and outputs $Q_a(v)$ in time $\text{poly}(h)$.*

Proof. Let $Q_0 = P$. For $a \in \{1, \ldots, m\}$ we recursively define

$$Q_a(v_1, \ldots, v_{a-1}, \underline{v_a}, v_{a+1}, \ldots, v_m) := \sum_{u \in H} \delta_u(v_a) \cdot Q_{a-1}(v_1, \ldots, v_{a-1}, \underline{u}, v_{a+1}, \ldots, v_m) \,,$$

where the underline above is added only to highlight the important part (i.e., reducing the individual degree of the a-th variable). The DSR condition follows from the definition and from the properties of δ_u. By induction, for any $a \in [m]$ the degree of any of the variables $v_1, \ldots, v_a$ in Q_a is $h-1$, and Q_a agrees with Q_{a-1} on H^m. ■

Proof of Proposition 6.4. Having the above tools, we can finally construct our list of polynomials as follows.

Polynomial Decomposition of $C_N(x)$

- Let $P_1^{(x)}$ be defined as in Lemma 6.5, which has maximum individual degree at most $h-1$ and agrees with $\alpha_1^{(x)}$ on H^m.
- For $i \in \{2, \ldots, d\}$, we start from $P_{i-1}^{(x)}$ that has maximum individual degree at most $h-1$ and agrees with $\alpha_{i-1}^{(x)}$ on H^m. Then, corresponding to layer i, we obtain a list of polynomials $(P_{i,0}^{(x)}, \ldots, P_{i,m}^{(x)})$ as follows.
  - First, applying Lemma 6.6, we lift $P_{i-1}^{(x)}$ to get $P_{i,0}^{(x)}$, which agrees with $\alpha_i^{(x)}$ on H^m and has total degree at most Δ'.
  - Then, we apply Lemma 6.7 to $P_{i,0}^{(x)}$ to obtain $P_{i,1}^{(x)}, \ldots, P_{i,m}^{(x)}$ that all agree with $\alpha_i^{(x)}$ on H^m, where $P_{i,a}^{(x)}$ has total degree at most Δ' for $a < m$, and $P_i^{(x)} := P_{i,m}^{(x)}$ has maximum individual degree $h-1$.
- The final list of polynomials is the list $P_{i,j}^{(x)}$ ordered by the lexicographic order on the indices (i,j).

We now show that the construction satisfies the claims in our statement. The degree bound holds since the total degrees of the P_i's is $m \cdot (h-1) \leq \Delta'$ and the total degrees of the $P_{i,j}$'s is at most Δ'. The algorithm for the first layer $P_1^{(x)}$ is the one from Lemma 6.5. The final layer algorithm follows since each $P_i^{(x)}$ agrees with $\alpha_i^{(x)}$ on H^m (by induction, using Lemma 6.6), and hence $P_d^{(x)}$ agrees with the output layer of C_N on H^m; given $j \in [w]$, the algorithm queries $P_d^{(x)}(\eta(j))$ and outputs the result, in time $\mathrm{poly}(h, \log(|\mathbb{F}|)) = \mathrm{poly}(h)$.

To show DSR, let $v \in \mathbb{F}^m$ and $P_{i,j}^{(x)}$ be given. If $j = 0$, this is the earliest layer $i+1$ polynomial and by DSR in Lemma 6.6, $P_{i,j}^{(x)}(v)$ is computable in time $\tilde{O}(T) + k \cdot \mathrm{poly}(h)$ using k queries to $P_{i-1}^{(x)}$. Otherwise, $P_{i,j}^{(x)}$ is a degree reduction polynomial, and by Lemma 6.7 it is computable in time $\mathrm{poly}(h)$ using h queries to $P_{i,j-1}^{(x)}$. So, $\max\{k, h\}$ queries and $\tilde{O}(T) + k \cdot \mathrm{poly}(h)$ time bound both cases.

Finally, we analyze the time required for outputting the truth tables of all polynomials. We have $O(d \cdot m)$ polynomials, each over domain of size $|\mathbb{F}|^m$. The first polynomial can be printed in time

$$w \cdot \tilde{O}(T) + |\mathbb{F}|^m \cdot (w \cdot \mathrm{poly}(h)) \leq |\mathbb{F}|^m \cdot \mathrm{poly}(h) \cdot \max\{w, \tilde{O}(T)\}$$

where we used the fact that $w \leq |\mathbb{F}|^m$. Then, using DSR, all other remaining polynomials can be computed in time

$$O(d \cdot m) \cdot |\mathbb{F}|^m \cdot (\tilde{O}(T) + k \cdot \mathrm{poly}(h)) \leq |\mathbb{F}|^m \cdot \mathrm{poly}(h) \cdot d \cdot \tilde{O}(T)\,,$$

where we used the facts that $k \leq T$ and $m \leq \log(w) \leq h$.

6.1.2 The reconstructive generator

We now present the proof of Theorem 6.3. The construction will rely on the following (variant of a) reconstructive generator.

Proposition 6.8 (classical generator for low-degree polynomials with a learning algorithm as the reconstruction algorithm.)**.** *There exists a universal constant c_0 such that the following holds. Let $\eta, \mu : \mathbb{N} \to (0,1)$ and $m, \Delta : \mathbb{N} \to \mathbb{N}$ be time-computable functions and let $\mathbb{F} = \mathbb{F}(n)$ be a family of finite fields with characteristic 2. Then, there exists a deterministic algorithm Gen_0 and a randomized oracle algorithm Rec_0 such that the following holds.*[38]

- ***Generator.*** *Gen_0 takes as inputs the parameters $|\mathbb{F}|, m, \mu$ and the (binary) truth-table of a polynomial $P : \mathbb{F}^m \to \mathbb{F}$* [39] *of degree $\Delta(n)$, runs in time $|\mathbb{F}|^{(1+c_0 \cdot \mu)(m+1)}$, and outputs a list L of binary strings of length $|\mathbb{F}|^{\mu(m+1)}$.*
- ***Reconstruction.*** *Given the parameters $|\mathbb{F}|, m, \eta, \mu$ and query access to the polynomial P as well as a distinguisher D for L, the algorithm Rec_0 runs in time $(|\mathbb{F}|^{1+\mu(m+1)} \cdot m \cdot \log 1/\eta)^{c_0}$ and outputs an oracle circuit C that acts as follows. Assuming D is a $(1/|\mathbb{F}|^{\mu(m+1)})$-distinguisher for L, and P has degree at most $\Delta \leq |\mathbb{F}|^{1-c_0 \cdot \mu(m+1)}$, with probability at least $(1-\eta)$ over the randomness of Rec_0 it holds that $C^D(v) = P(v)$ for all $v \in \mathbb{F}^m$.*

The proof of Proposition 6.8 follows the rather standard construction: at a high level, the generator Gen_0 applies the concatenation of the Nisan-Wigderson generator [NW94] and the Hadamard transformation to the truth-table of the polynomial P, and outputs a list of strings L. Then, given query access to P and a distinguisher D for L, the reconstruction algorithm Rec_0 first obtains a list of possible candidates for computing $P(\cdot)$ by repeatedly applying the concatenation of Nisan-Wigderson reconstruction algorithm [NW94], the local list-decoder for the Hadamard code [GL89], and the local list-decoder for Reed-Muller code [STV01]. Then, it "weeds" these candidates by probabilistically checking random samples $(v, P(v))$ from the oracle $P(\cdot)$. Finally, it applies a unique decoder of Reed-Muller [GLR+91] on the best remaining candidate.

Setting. Let c be a large enough constant whose value will be determined as we go through the proof. For $x \in \{0,1\}^N$, let $\{P_i^{(x)}\}_{i \in [d']}$ be the polynomial decomposition of Proposition 6.4 with parameters $h = w^\epsilon$ and $|\mathbb{F}| = (\Delta \cdot h)^{2+\epsilon}$.[40] Then, $m = \log w / \log h = 1/\epsilon$, and $d' = O(d/\epsilon)$, and $|\mathbb{F}|^m = \Delta^{(2+\epsilon)/\epsilon} \cdot w^{2+\epsilon}$.

[38] It helps to intuitively think of $\mu(m+1) \approx \epsilon$ being a small constant.

[39] By truth table of $P : \mathbb{F}^m \to \mathbb{F}$, we mean the truth table of the canonical binary representation $p : \{0,1\}^{m \cdot \log |\mathbb{F}|} \to \{0,1\}^{\log |\mathbb{F}|}$.

[40] For simplicity, we assume that $(\Delta \cdot h)^{2+\epsilon}$ is a power of two; a multiplicative increase of at most 2 in the size of $\mathbb{F}$ will not affect our argument. Note that a field of such size can be found in time $\mathrm{polylog}(|\mathbb{F}|)$.

Let c_0 be the universal constant in Proposition 6.8, and let Gen_0 and Rec_0 be the generator and reconstruction from Proposition 6.8 with parameters $\mu := \frac{\epsilon}{4c_0(m+1)}$ and $\eta := \frac{1}{c \cdot d'}$.

Generator. Given input $x \in \{0,1\}^{n+r(n)}$, the generator computes the truth-tables of all the polynomials $\{P_i^{(x)}\}_{i\in[d']}$, applies Gen_0 to the truth-table of each $P_i^{(x)}$ to obtain a corresponding list $L_i^{(x)} \subset \{0,1\}^{|\mathbb{F}|^{\mu(m+1)}}$, and outputs $L^{(x)} = \bigcup_{i\in[d']} L_i^{(x)}$.

Generator complexity analysis. By Proposition 6.8, the output length is $M = |\mathbb{F}|^{\mu(m+1)} = (\Delta^{1/\epsilon} \cdot w)^{\Omega(\epsilon^2)}$, and each execution of Gen_0 takes time

$$\begin{aligned} |\mathbb{F}|^{(1+c_0\mu)(m+1)} &= |\mathbb{F}|^m \cdot |\mathbb{F}| \cdot |\mathbb{F}|^{c_0\mu(m+1)} \\ &\leq \Delta^{(2+\epsilon)/\epsilon} \cdot w^{2+\epsilon} \cdot (\Delta \cdot h)^{2+\epsilon} \cdot \mathrm{poly}(\Delta^\epsilon, w^{\epsilon^2}) \\ &< \Delta^{3/\epsilon} \cdot w^{2+O(\epsilon)} \ . \end{aligned}$$

Repeating this for $d' = O(d/\epsilon)$ times, the total runtime of applying Gen_0 to all $d' = O(d/\epsilon)$ polynomials is at most $d \cdot \Delta^{3/\epsilon} \cdot w^{2+O(\epsilon)}$. This expression also upper bounds number of output strings.

For the total running time, we note that by Proposition 6.4, the the truth-table of all polynomials can be printed in time

$$d \cdot |\mathbb{F}|^m \cdot \mathrm{poly}(h) \cdot \max\left\{w, \tilde{O}(T)\right\} < d \cdot \Delta^{3/\epsilon} \cdot w^{2+O(\epsilon)} \cdot \max\left\{w, \tilde{O}(T)\right\} \ ,$$

and this dominates the runtime of the generator.

Reconstruction. The reconstruction algorithm works as follows.

Reconstruction algorithm Rec

- **Input:** The input x and query access to $D : \{0,1\}^M \to \{0,1\}$.
- **First layer:** Run Rec_0^{D,P_1} from Proposition 6.8, responding to each query $v \in \mathbb{F}^m$ that Rec_0 makes to P_1 by running the first layer algorithm $\mathrm{Base}(x, v)$ from Proposition 6.4. The algorithm Rec_0 returns an oracle circuit $\tilde{C}_1^D$.
- **Upper layers:** For $i \in \{2, \ldots, d'\}$, we start with an oracle circuit $\tilde{C}_{i-1}^D$. To obtain $\tilde{C}_i^D$, we run Rec_0^{D,P_i} of Proposition 6.8, responding to each query $v \in \mathbb{F}^m$ that Rec_0 makes to P_i by running $\mathrm{DSR}^{\tilde{C}_{i-1}^D}(i, v)$ of Proposition 6.4.
- **Output.** Given $\tilde{C}_{d'}^D$, we use the final layer algorithm of Proposition 6.4 to output the string $\bigcirc_{j\in[w]} \mathsf{Final}^{\tilde{C}_{d'}^D}(j)$.

Correctness of the reconstruction. Let $D : \{0,1\}^M \to \{0,1\}$ such that for every $i \in [d']$ we have $|\mu_D - \mathbb{E}_{y \sim L_i^{(x)}}[D(y)]| \geq 1/M$, where $\mu_D = \mathbb{E}_{y \sim \{0,1\}^M}[D(y)]$. We prove that with high probability the reconstruction outputs $C_N(x)$.

By Proposition 6.4, we know that the total degree of each $P_i^{(x)}$ is at most $\Delta' = \Delta \cdot h^2 \cdot \text{polylog}(w)$. Since $|\mathbb{F}| = \Delta^{(2+\epsilon)/\epsilon} \cdot h^{2+\epsilon}$, the degree requirement in the hypothesis of Proposition 6.8 is satisfied; that is,

$$|\mathbb{F}|^{1-c_0 \cdot \mu(m+1)} = |\mathbb{F}|^{1-\epsilon/4c_0} > \Delta \cdot h^{(2+\epsilon)(1-\epsilon/4c_0)} > \Delta \cdot h^2 \cdot \text{polylog}(w) \, .$$

Using Proposition 6.8, our hypothesis that D is a $(1/M)$-distinguisher for each $L_i^{(x)}$, and induction on $i \in [d']$, with probability at most $(1 - d' \cdot \eta) = 1 - 1/c \geq 0.9$ the circuit $\tilde{C}_{d'}^D$ correctly computes $P_{d'}^{(x)}$, in which case $\mathsf{Final}^{\tilde{C}_{d'}^D}(j) = (C_N(x))_j$ for all $j \in [w]$.

Complexity analysis. By Proposition 6.8, each execution of Rec_0^{D,P_i} takes time

$$|\mathbb{F}|^{1+\mu(m+1)} \leq \text{poly}(\Delta, w^\epsilon) \, ,$$

and therefore, each circuit $\tilde{C}_i^D$ for $i \in [d']$ also has size $\text{poly}(\Delta, w^\epsilon)$. For the first layer, the total running time is upper bounded by

$$\underbrace{\text{poly}(\Delta, w^\epsilon)}_{\mathsf{Rec}_0 \text{ time}} \cdot \underbrace{(w \cdot (\tilde{O}(T) + \text{poly}(h)))}_{\text{Base time}} = \text{poly}(\Delta) \cdot w^{1+O(\epsilon)} \cdot \tilde{O}(T) \, .$$

For any of the upper layers $i \in \{2, \ldots, d'\}$, we need to account for the runtime of Rec_0, of DSR, and also of answering each of the $\max\{k,h\}$ queries that DSR makes by running $\tilde{C}_{i-1}^D$. The running time of each iteration i is thus

$$\underbrace{\text{poly}(\Delta, w^\epsilon)}_{\mathsf{Rec}_0 \text{ time}} \cdot \underbrace{\left(\tilde{O}(T) + k \cdot \text{poly}(h)\right)}_{\text{DSR time}} + \underbrace{\max\{k,h\} \cdot \text{poly}(\Delta, w^\epsilon)}_{\tilde{C}_{i-1}^D \text{ time}} \leq \text{poly}(\Delta, w^\epsilon) \cdot \max\left\{k, \tilde{O}(T)\right\}$$

Printing the last layer using Final also takes at most $w \cdot \text{poly}(\Delta, w^\epsilon)$. Therefore, the total running time is bounded by

$$(w + d) \cdot \text{poly}(\Delta, w^\epsilon) \cdot \max\left\{k, \tilde{O}(T)\right\} \, .$$

It only remains to bound the total number of queries to D. Recall that Rec_0 and $\tilde{C}_i^D$ make at most $Q = \text{poly}(\Delta, w^\epsilon)$ queries to D. In the first layer, we make at most Q queries by running Rec_0. For the upper layers $i \in \{2, \ldots, d'\}$, the algorithm DSR makes at most $(k + h) < k \cdot w^\epsilon$ many queries to $\tilde{C}_{i-1}^D$, so the total number of queries is

$$\underbrace{Q}_{\mathsf{Rec}_0 \text{ queries to } D} + O(d) \cdot \underbrace{Q}_{\mathsf{Rec}_0 \text{ time}} \cdot \underbrace{k \cdot w^\epsilon}_{\text{DSR queries to } \tilde{C}_{i-1}^D} \cdot \underbrace{Q}_{\tilde{C}_{i-1}^D \text{queries to } D} \leq d \cdot k \cdot \text{poly}(\Delta, w^\epsilon) \, .$$

Finally, printing the final layer makes $w \cdot Q$ queries to D, and so the total number of queries is at most $(d \cdot k + w) \cdot \text{poly}(\Delta, w^\epsilon)$. ∎

6.2 The targeted hitter and error-reduction

The targeted hitter of Theorem 1.9 is the targeted generator of Theorem 6.3, instantiated with appropriate parameters. We first state a more general version and then deduce Theorem 1.9 as a corollary.

Theorem 6.9 (targeted hitter based on hardness of uniform circuits with low-degree gates)**.** *There exists a universal constant c such that the following holds. Let $T, r, w, d : \mathbb{N} \to \mathbb{N}$ be time-constructible such that w always being a power of 2 and $T(n) \geq \max\{d(n), w(n)\}$, and let $\bar{T} := w(n) \cdot T(n) \geq n + r(n)$. Also, let $\epsilon \in (0,1)$ be a small enough constant, and let $\delta : \mathbb{N} \to (0,1)$. Assume that there is a highly uniform family $\left\{C_{n+r(n)} \colon \{0,1\}^{n+r(n)} \to \{0,1\}^{w(n)}\right\}$ of $(d \times w)$-circuits with $(T^{\epsilon^2}, \bar{T})$-low-degree gates such that the following holds.*[41] *For every randomized algorithm A running in time $(w+d) \cdot w^{c\cdot\epsilon} \cdot T^{c\cdot\epsilon^2} \cdot \tilde{O}(\bar{T})$, for all but finitely many inputs $x_1 \in \{0,1\}^n$, it holds that*

$$\Pr_{x_2 \in \{0,1\}^{r(n)}}\left[\Pr_A[A(x_1 \circ x_2) = C_{n+r(n)}(x_1 \circ x_2)] \geq 2/3\right] < \delta\,.$$

Then, there exists a $(w^{-\Omega(\epsilon^2)}, \delta)$-targeted hitter against T-time algorithms that runs in time $d \cdot w^2 \cdot T^{1+c\cdot\epsilon}$; and it has randomness complexity $r(n)$, sample complexity $d \cdot w^2 \cdot T^{c\cdot\epsilon}$, and output length $T^{\epsilon^3/c} \cdot w^{\epsilon^2/c}$.

Proof. Let c be a large enough constant whose value will be determined as we go through the proof, let c' be the constant in Theorem 6.3, and let $\epsilon > 0$ be sufficiently small. Also, let Gen and Rec be the pair of deterministic and randomized algorithms obtained when instantiating Theorem 6.3 with the circuit family $\{C_{N=n+r(n)}\}$, with a trivial bound of w on the fan-in of gates above the first layer.

Targeted hitter. We define our hitter $H(x, r = x_2)$ to output the union of lists $L^{(x_1\circ x_2)} = \bigcup_i L_i^{(x_1\circ x_2)}$ outputted by $\mathsf{Gen}(x_1 \circ x_2)$. The output length of H is $M = T^{\Omega(\epsilon^3)} \cdot w^{\Omega(\epsilon^2)} \geq T^{\epsilon^3/c} \cdot w^{\epsilon^2/c}$, and printing all the sampled strings takes time at most

$$d \cdot T^{3\epsilon} \cdot w^{2+O(\epsilon)} \cdot \tilde{O}(\bar{T}) < d \cdot w^2 \cdot T^{1+c\epsilon},$$

and the sample complexity is at most

$$S = |L^{(x)}| \leq d \cdot T^{3\epsilon} \cdot w^{2+O(\epsilon)} < d \cdot w^2 \cdot T^{c\epsilon}.$$

Correctness. Assume towards a contradiction that there exists a T-time $(1/M, \delta)$-distinguisher $D : \{0,1\}^n \times \{0,1\}^M \to \{0,1\}$ for H; that is, for infinitely many inputs $x_1 \in \{0,1\}^n$ we have $\mathbb{E}_{y\sim\{0,1\}^M}[D(x_1, y)] \geq 1/M$, but for at least a δ-fraction of $x_2 \in \{0,1\}^{r(n)}$ we

[41] Indeed, in this statement there is no fan-in restriction on the gates. Also, as in Theorem 6.3, the condition $w(n) \cdot T(n) \geq n + r(n)$ is not actually necessary, and we include it to assist with the parsing of the parameters (it ensures that the first layer of the circuit has enough wires to read all the input bits).

have $\mathbb{E}_{y\sim L^{(x_1\circ x_2)}}[D(x_1,y)] = 0$. Then, by Theorem 6.3, for any such x_1 and a δ-fraction of $x_2 \in \{0,1\}^{r(n)}$, the algorithm $A(x_1 \circ x_2) := \mathsf{Rec}^D(x_1 \circ x_2)$ computes $C_{n+r(n)}(x_1 \circ x_2)$ with probability at least $0.9 > 2/3$.

By Theorem 6.3, assuming $c > c'$ is large enough, Rec^D runs in time

$$(w+d)\cdot \mathrm{poly}(T^{\epsilon^2}, w^{\epsilon})\cdot \tilde{O}(\bar{T}) < (w+d)\cdot T^{c\cdot\epsilon^2}\cdot w^{c\cdot\epsilon}\cdot\tilde{O}(\bar{T})$$

and the number of queries it makes to D is at most

$$(T^{\epsilon^2}\cdot w^{\epsilon})^{c'}\cdot(w+d\cdot w) \le w^{1+c\cdot\epsilon}\cdot d\cdot T^{c\cdot\epsilon^2}\ .$$

Resolving each query in time T, the total runtime of A is

$$\begin{aligned}
&(w+d)\cdot T^{c\cdot\epsilon^2}\cdot w^{c\cdot\epsilon}\cdot\tilde{O}(\bar{T}) + w^{1+c\cdot\epsilon}\cdot d\cdot T^{c\cdot\epsilon^2}\cdot T\\
&\le \left((w+d)\cdot T^{c\cdot\epsilon^2}\cdot w^{c\cdot\epsilon} + w^{c\cdot\epsilon}\cdot d\cdot T^{c\cdot\epsilon^2}\right)\cdot\tilde{O}(\bar{T})\\
&\le (w+d)\cdot w^{c\cdot\epsilon}\cdot T^{c\cdot\epsilon^2}\cdot\tilde{O}(\bar{T})\ ,
\end{aligned}$$

contradicting the hardness assumption. ■

Theorem 1.9 follows by instantiating Theorem 6.9 with parameter values $k = w = d = T^{O(\epsilon)}$ and $r(n) = O(T(n))$. We state the theorem without assuming OWFs and with small output length; using Theorem 3.3, if we additionally assume OWFs we can increase the output length to T without meaningfully increasing the running time (and with no effect on the randomness or sample size).

Corollary 6.10 (Theorem 1.9, restated). *Let $T\colon \mathbb{N}\to\mathbb{N}$ be time-constructible, let $\epsilon,\alpha>0$ be constants, and let $k = T^{O_\alpha(\epsilon)}$. For $\bar{T} = T^{1+O(\epsilon)}$, assume that there is a highly uniform family $\left\{C_{n+O(T(n))}\colon \{0,1\}^{n+O(T(n))}\to\{0,1\}^k\right\}$ of $(k\times k)$-circuits with $(T^{\epsilon^2},\bar{T})$-low-degree gates such that the following holds: For every randomized algorithm A running in time $k^{1+\alpha}\cdot\bar{T}$, for all but finitely many $x_1\in\{0,1\}^n$,*

$$\Pr_{x_2\in\{0,1\}^{O(T(n))}}\left[\Pr_A[A(x_1\circ x_2) = C_{n+O(T(n))}(x_1\circ x_2)]\ge 2/3\right] < \delta\ .$$

Then, there exists a $(T^{-\Omega(\epsilon)},\delta)$-targeted hitter against T-time algorithms that runs in time $T^{1+O(\epsilon)}$ and has randomness $O(T(n))$, sample complexity $T^{O(\epsilon)}$, and output length $T^{\Omega(\epsilon^2)}$.

7 Targeted Hitters Based on Hardness of Probabilistic Levin-Kolmogorov Complexity

In this section we construct a targeted hitter that is based on hardness of estimating probabilistic time-bounded Kolmogorov complexity, and use it to deduce a conditional error reduction result for probabilistic algorithms with one-sided error.

We first define fundamental meta-complexity concepts, then describe the construction of our targeted hitting-set generator (Proposition 7.4), and finally we apply the generator as a targeted hitter to obtain error reduction for prRTIME.

Preliminaries. Throughout the section we fix some fixed Universal Turing machine U that can emulate any Turing machine Π with polylogarithmic time overhead. Denote by $U(\Pi(z), 1^t)$ the output of $\Pi(z)$ when emulated on U for t steps if it halts. If $\Pi(z)$ does not halt within t steps, then $U(\Pi(z), 1^t)$ outputs $\perp$.

We will rely on the notions of probabilistic time-bounded Levin-Kolmogorov complexity as introduced by Goldberg *et al.* [GKL+22], and in particular on the conditional variant cpKt and the GapMcpKtP problem, defined as follows.

Definition 7.1 (cpKt). *For every $\delta \in (0,1)$, $w \in \{0,1\}^*$, and $x \in \{0,1\}^*$, the conditional probabilistic Levin-Kolmogorov complexity of w given advice string x and confidence δ is defined as*

$$\mathsf{cpKt}_\delta(w|x) := \min_\ell \left\{ \ell \in \mathbb{N} : \Pr_{r \in \{0,1\}^{2^\ell}} \left[\exists \Pi \in \{0,1\}^{\leq \ell} : U(\Pi(x;r), 1^{2^{\ell - |\Pi|}}) = w \right] \geq \delta \right\}.$$

Definition 7.2 (GapMcpKtP). *For every real function $\delta : \mathbb{N} \to (0,1)$ and $m, T_Y, T_N : \mathbb{N} \to \mathbb{N}$, the promise problem $\mathsf{GapMcpKtP}_\delta[T_Y, T_N] = (\Pi_Y, \Pi_N)$ with length $m(n)$ is defined as follows.*

- *YES instances Π_Y: (w,x) such that $|w| = m(|x|)$ and $\mathsf{cpKt}_{\delta(|w|)}(w|x) \leq T_Y(|w|)$.*
- *NO instances Π_N: (w,x) such that $|w| = m(|x|)$ and $\mathsf{cpKt}_{\delta(|w|)}(w|x) \geq T_N(|w|)$.*

For a fixed auxiliary string x, we say that an algorithm M decides *the slice of $\mathsf{GapMcpKtP}_\delta[T_Y, T_N]$ on x with length m, if for every $w \in \{0,1\}^{m(|x|)}$,*

$$\begin{cases} M(w;x) = 1, & \text{if } \mathsf{cpKt}_\delta(w|x) \leq T_Y(|w|), \\ M(w;x) = 0, & \text{if } \mathsf{cpKt}_\delta(w|x) \geq T_N(|w|). \end{cases}$$

7.1 A reconstructive targeted generator

The construction of the targeted hitting-set generator will rely on the following coding theorem for cpKt, which is a straightforward adaptation of the coding theorem of [LOZ22]. For completeness, we include a proof in Appendix C.2.

Lemma 7.3 (coding theorem for cpKt, following [LOZ22]; see Appendix C.2). *Suppose there is a randomized algorithm A for sampling strings such that $A(1^n; x)$ runs in time $T(n) \geq n$ and outputs a string $w \in \{0,1\}^n$ with probability at least $\delta > 0$. Then for every $0 < \beta \leq 1$,*

$$\mathsf{cpKt}_\beta(w|x) \leq \log(1/\delta) + O(\log T(n)) + O(\log\log 1/(1-\beta)) + O_{|A|}(1),$$

and the constants behind $O(\log T(n))$ and $\mathrm{poly}(T)$ are independent of A.

We present the targeted hitting-set generator with more general parameters than in Theorem 1.11. The parameters of the latter result follows by taking $k(T) = T^\epsilon$, $\ell(n) = (1+\epsilon)\log n$ and $\delta(m) = 2^{-\Omega(m^{1+\epsilon})}$.

Proposition 7.4 (targeted hitting-set generator based on hardness of approximating cpKt)**.** *For any increasing time-constructible functions $T, k, \ell : \mathbb{N} \to \mathbb{N}$ satisfying*

$$T(n) \geq n, \quad k(T) = \Omega(\log T), \quad \ell(|w|) \geq \log |w| + \mathrm{loglog}|w|, \quad \ell(k(T) \cdot T) \leq O(k(T)),$$

there exists a generator H that satisfies the following. For every linear-time deterministic algorithm D and $\delta : \mathbb{N} \to (0,1)$ satisfying $2^{-2^{\ell(m)}} < \delta(m) < 1/2$, and every sufficiently large $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, the following holds.

- ***Generator.*** *The generator H gets input x and $r \in \{0,1\}^L$, outputs a list of strings of length T in time $\tilde{O}(L)$, and the list has size at most $O(L/T)$, where $L = 2^{\ell(kT)}$ and $T = T(n)$ and $k = k(T)$.*
- ***Distinguisher.*** *If the linear-time deterministic algorithm $D(x; z)$ with $|z| = T(|x|)$ satisfies*

$$\Pr[D(x; U_T) = 1] \geq 1/2,$$
$$\Pr_r[\forall z \in H(x; r) : D(x; z) = 0] > \delta(kT),$$

then the linear-time deterministic algorithm $A(w; x) := \bigwedge_{i \in [k]} \neg D(x; w_i)$ decides the slice of $\mathsf{GapMcpKtP}_{1-\delta}[\ell(|w|), |w| - \Theta(k)]$ on x with length $|w| = kT$, where w_i is defined by partitioning w into k substrings $w_1 w_2 \ldots w_k = w$ of length T.

Proof. We define $\mathcal{M}_m$ to be the set of all Turing machines with description size at most m (with respect to the universal Turing machine $\mathcal{U}$ that we fixed).

The generator H. Given input $x \in \{0,1\}^n$ and randomness $r \in \{0,1\}^L$, the generator computes the set $S(x; r)$ of length-kT strings w such that $\mathsf{cKt}(w|x, r) \leq \ell(kT)$; see below on how it compute $S(x; r)$. Then, for each $w \in S(x; r)$, the generator divides w into k disjoint strings $w_1, w_2, \ldots, w_k$ of length T and outputs the k strings.

In more detail, note that if a program M outputs a string of length kT, its running time should be at least kT, which implies that

$$2^{\ell(kT) - |M|} \geq kT \implies |M| \leq \ell(kT) - \log kT =: \ell'.$$

Therefore, to compute $S(x; r)$, it suffices for the generator to enumerate over all programs of description length at most ℓ' (and simulate each machine of description length i on input (x, r) for $2^{\ell(kT)-i}$ steps).

Consequently, the size of output list is at most $k \cdot |S(x; r)| \leq k \cdot |\mathcal{M}_{\ell'}| = O(L/T)$. The running time of H, including simulating each machine and post-processing the list in time linear in kT, is at most

$$\sum_{i \in [\ell']} 2^i \cdot \tilde{O}(2^{\ell(kT)-i}) + O(kT) = \ell' \cdot \tilde{O}(L) + O(kT) = \tilde{O}(L) \,.$$

The distinguisher D. For a fixed x, suppose that D satisfies

$$\Pr[D(x;U_T)=1] \geq 1/2,$$
$$\Pr_r[\forall z \in H(x;r) : D(x;z)=0] > \delta(kT).$$

Let $A(w;x) := \bigwedge_{i\in[k]} \neg D(x;w_i)$. Note that the running time of A is $O(k \cdot T)$ (since D runs in time linear in its input length $O(T)$), i.e. A runs in linear time in its input length. We will argue that $A(w;x)$ decides the slice of $\mathsf{GapMcpKtP}_{1-\delta}[\ell(|w|), |w| - \Theta(k)]$ on x, using the following two claims.

Claim 7.4.1. *$A(\cdot;x)$ accepts every $w \in \{0,1\}^{kT}$ satisfying $\mathsf{cpKt}_{1-\delta}(w|x) \leq \ell(|w|)$.*

Proof. By the definition of $\mathsf{cpKt}_{1-\delta}$, there exists a set of strings $R \subseteq \{0,1\}^L$ of size at least $(1-\delta(kT))2^L$ such that

$$\forall r \in R : \exists M_r \in \mathcal{M}_{\ell(kT)} : U(M_r(x;r); 1^{2^{\ell(kT)-|M_r|}}) = w,$$

For each $r \in R$, we know that

$$S(x;r) \ni U(M_r(x;r); 1^{2^{\ell(kT)-|M_r|}}) = w.$$

Since $D(x;\cdot)$ rejects $H(x;r)$ with probability greater than $\delta(kT)$ over r, we know that there exists an $r \in R$ such that $D(x;\cdot)$ rejects all the strings in $H(x;r)$, including $w_1, w_2, \ldots, w_k$. Therefore, $A(\cdot;x)$ accepts w. □

Claim 7.4.2. *$A(\cdot;x)$ rejects every $w \in \{0,1\}^{kT}$ satisfying $\mathsf{cpKt}_{1-\delta}(w|x) > |w| - \Theta(k)$.*

Proof. Assuming that $A(w;x) = 1$ for some w, we show that $\mathsf{cpKt}_{1-\delta}(w|x) \leq |w| - \Theta(k)$.

First, since $A(w;x) = 1$, we deduce that $D(x;w_i) = 0$ for all $i \in [k]$. Now, consider the following linear-time randomized sampler $B(1^N;x)$ for w.

1. Calculate $T = T(|x|)$ and $k = N/T$.
2. For each $i \in [k]$, sample $z_{i,0}, z_{i,1} \sim \{0,1\}^T$.
3. If $D(x;z_{i,0}) = 0$, then let $w_i = z_{i,0}$; otherwise let $w_i = z_{i,1}$.
4. Output $w = w_1 w_2 \ldots w_k$.

For each $i \in [k]$, the probability that $B(1^{kT};x)$ outputs w_i correctly is

$$\Pr_r[r = w_i] \cdot (1 + \Pr_r[D(x;r)=1]) \geq 2^{-T} \cdot (1 + 1/2),$$

and the probability that $B(1^{kT};x)$ outputs w is

$$\Pr[B(1^{kT};x) = w] \geq (1.5 \times 2^{-T})^k = 1.5^k \cdot 2^{-|w|}.$$

Applying Lemma 7.3, we deduce that

$$\begin{aligned}\mathsf{cpKt}_{1-\delta}(w|x) &\leq |w| - \Theta(k) + O(\log|w|) + O(\log\log(1/\delta)) + O_{|B|}(1)\\ &\leq |w| - \Theta(k) + O(\log|w|) + O(\ell(kT)) + O_{|B|}(1)\\ &\leq |w| - \Theta(k).\end{aligned}$$

□

Combining the two foregoing claims, we conclude that $A(\cdot; x)$ decides the slice of $\mathsf{GapMcpKtP}_{1-\delta}[\ell(|w|), |w| - \Theta(k)]$ on x with length kT in linear time. ■

As a direct corollary of Proposition 7.4, we obtain the following targeted hitter.

Corollary 7.5 (targeted hitter based on hardness of estimating cpKt; Theorem 1.11, restated with general parameters)**.** *Let $T, k, \ell : \mathbb{N} \to \mathbb{N}$ be monotone time-constructible functions satisfying*

$$T(n) \geq n, \quad k(T) = \Omega(\log T), \quad \ell(|w|) \geq \log|w| + \mathrm{loglog}|w|, \quad \ell(k(T)\cdot T) = O(k(T)),$$

and let $\delta : \mathbb{N} \to (0,1)$ be such that $2^{-2^{\ell(m)}} < \delta(m) < 1/2$ for sufficiently large m.

Assume that any linear-time algorithm $A(w; x)$ fails to decide the slice of $\mathsf{GapMcpKtP}_{1-\delta}[\ell(|w|), |w| - \Theta(k)]$ on x with length $|w| = kT$ for all but finitely many x, where $k = k(T)$, $T = T(n)$, and $\delta = \delta(kT)$. Then, there exists a $(1/2, \delta(kT))$-targeted hitter against T-time algorithms that has randomness complexity $L = 2^{\ell(kT)}$, sample complexity $O(L/T)$, and output length T, and runs in time $\tilde{O}(L)$.

Note that the parameter sestting in Corollary 7.5 is not transparent, since δ is a function of kT but the randomness to the hitter is of length $L = 2^{\ell(kT)}$. Theorem 1.11 follows by instantiating Corollary 7.5 with $k(T) = T^{\epsilon}$, $\ell(n) = (1+\epsilon)\log n$ and $\delta(m) = 2^{-\Omega(m^{1+\epsilon})}$ (such that $\delta(kT) = 2^{-\Omega(kT^{1+\epsilon})} = 2^{-\Omega(L)}$).

7.2 The targeted hitter and error-reduction

We now spell out the conditional error-reduction results obtained using the targeted hitter construction from Corollary 7.5. These are straightforward consequences, and we flesh out the details because the parameter settings may not be immediately obvious without seeing the detailed proofs below.

Theorem 7.6 (error-reduction using the targeted hitter)**.** *For every monotone time-constructible functions $T, k, \ell : \mathbb{N} \to \mathbb{N}$ satisfying*

$$T(n) \geq n, \quad k(T) = \Omega(\log T), \quad \ell(|w|) \geq \log|w| + \mathrm{loglog}|w|, \quad \ell(k(T)\cdot T) = O(k(T)),$$

and real function $\delta : \mathbb{N} \to (0,1)$ such that $2^{-2^{\ell(m)}} < \delta(m) < 1/2$ for sufficiently large m, the following holds.

For simplicity, we write $n = |x|$, $T = T(n)$ and $k = k(T)$. Let $L = 2^{\ell(kT)}$. Suppose every linear-time algorithm $A(w;x)$ decides the slice of $\mathsf{GapMcpKtP}_{1-\delta}[\ell(|w|), |w| - \Theta(k)]$ on x with length $|w| = kT$ for only finitely many x. Then

$$\mathsf{prRTIME}_{1/2}[T] \subseteq \mathsf{prRTC}_{2^L \cdot \delta(\ell^{-1}(\log L))}[\tilde{O}(L), L],$$

where ℓ^{-1} is the inverse function of ℓ.

Proof. Let H be the $(1/2, \delta(kT))$-targeted hitter of Corollary 7.5 initialized with the same parameters T, k, ℓ, and δ. For $\Pi = (\Pi_Y, \Pi_N) \in \mathsf{prRTIME}_{1/2}[T]$ decided by a T-time randomized algorithm $D(x;z)$, we define $B(x;r) := \bigvee_{z \in H(x;r)} D(x;z)$ for $r \in \{0,1\}^L$. We show that B decides Π with error probability at most $\delta(\ell^{-1}(\log |r|))$ over the randomness r on all but finitely many inputs in the promise.

Clearly B rejects every $x \in \Pi_N$ with probability 1. For $x \in \Pi_Y$, if $\Pr_r[B(x;r) = 1] < 1 - \delta$ then

$$\mathbb{E}_{z \in \{0,1\}^T}[D(x;z)] \geq 1/2,$$

whereas for at least δ fraction of $r \in \{0,1\}^L$,

$$\forall z \in H(x;r): \quad D(x,z) = 0.$$

Consequently, D is a T-time distinguisher for H on x for at least a δ fraction of $r \in \{0,1\}^L$, which contradicts the assumption that H is a $(1/2, \delta)$-targeted hitter. Therefore, for all but finitely many $x \in \Pi_Y$, $B(\cdot; r)$ accepts x with error probability at least $1 - \delta$ over r.

The running time of B is bounded by $\tilde{O}(L)$, since evaluating $H(x;r)$ takes $\tilde{O}(L)$ time and running D for $|H(x;r)| = O(L/T)$ rounds takes $O(L)$ time.The number of random coins is $|r| = L$, and the number of bad coins as a function of L is $2^L \cdot \delta(kT) = 2^L \cdot \delta(\ell^{-1}(\log L))$. ∎

The following corollary of Theorem 7.6 is obtained using the same parameter setting as explained after Corollary 7.5; that is, we set $k(T) = T^\epsilon$, and $\ell(n) = (1+\epsilon)\log n$ and $\delta(m) = 2^{-\Omega(m^{1+\epsilon})}$ (so that $\delta(kT) = 2^{-\Omega(L)}$).

Corollary 7.7 (error-reduction from hardness of estimating cpKt)**.** *For every monotone time-constructible function $T : \mathbb{N} \to \mathbb{N}$ satisfying $T(n) \geq n$ and constant $\epsilon > 0$, the following holds. Suppose that every linear-time algorithm $A(w;x)$ decides the slice of*

$$\mathsf{GapMcpKtP}_{1-2^{-\Omega(|w|^{1+\epsilon})}}\left[(1+\epsilon)\log|w|, |w| - \Theta(|w|^{\epsilon/(1+\epsilon)})\right]$$

on x with length $|w| = T^{1+\epsilon}(|x|)$ for only finitely many x. Then

$$\mathsf{prRTIME}_{1/2}[T] \subseteq \mathsf{prRTC}_{2^{(1-\Omega(1))C}}[T^{1+O(\epsilon)}, C := T^{1+O(\epsilon)}].$$

8 Codes with a Doubly-Local Decoder

In this section we present our codes with doubly-local decoders, and prove Theorems 1.13 and 1.14. Recall that the main goal is to (approximately) list-decode from exponentially small agreement δ in time $\mathrm{polylog}(1/\delta)$, where the decoder is allowed to use a non-deterministic oracle.

Let us first recall the standard definition of locally list-decodable codes. To be more consistent with the coding literature, we will at this point **switch notation** from the rest of the paper: we will denote the exponentially small agreement by ϵ (instead of δ), and reserve δ for the approximation parameter (i.e., the decoder succeeds on $1-\delta$ of the coordinates). Indeed, these roles of ϵ and δ are the opposite of their roles in the literature about samplers, which informs the rest of the paper. We also use n and N for the message length and block length, respectively.[42]

Definition 8.1 (locally list-decodable code)**.** *We say that a code $\mathcal{C}\colon \Sigma_1^n \to \Sigma_2^N$ is δ-approximate (ϵ, L) locally list decodable with Q queries if there exists a randomized algorithm $\mathcal{A}$ that (on input 1^N) outputs (randomized) circuits $C_1, \ldots, C_L$ such that for any $w \in \Sigma_2^N$, and $f \in \Sigma_1^n$ such that $\Pr_{i\in[N]}[\mathcal{C}(f)_i = w_i] \geq \epsilon$, it holds that*

$$\Pr_{\mathcal{A}}\left[\exists i \in [L] : \Pr_{x\in[n],C_i}[C_i^w(x) = f(x)] \geq 1-\delta\right] \geq \frac{3}{4}$$

Moreover, each C_i makes at most Q queries to w.

We are interested in the tiny-ϵ regime, namely $\epsilon = n^{-\Omega(1)}$, and we cannot afford to pay $\mathrm{poly}(1/\epsilon)$ in the number of queries, decoder size, and locality. Our main result, which is a technical version of Theorem 1.14, is stated below. Theorem 1.13 will be established as one of the components going into the proof below.

Corollary 8.2 (the special case of Theorem 8.6 for list-decoding)**.** *For any $n \in \mathbb{N}$, $\epsilon = \epsilon(n), \delta = \delta(n) \in (0,1)$, there exists an explicit code $\mathcal{C}\colon \{0,1\}^n \to \Sigma^{n'}$ that is δ-approximate (ϵ, L) locally list decodable with $\log|\Sigma| = O(\log(1/\epsilon))$, $L = \widetilde{O}(1/\epsilon)$, and*

$$n'/n = \left(\frac{\log n}{\delta}\right)^{O(\log^2(1/\epsilon))}$$

in the following sense.

There exists a randomized algorithm $\mathcal{A}$ that generates randomized oracle circuits $C_1, \ldots, C_L$ such that for any $w \in \Sigma^{n'}$, for every $f \in \{0,1\}^n$ that satisfies

$$\Pr_{i\in[n']}[\mathcal{C}(f)_i = w_i] \geq \epsilon,$$

it holds that

$$\Pr_{\mathcal{A}}\left[\exists i \in [L] : \Pr_{x\in[n],C_i}[D_i^w(x) = f(x)] \geq 1-\delta\right] \geq \frac{3}{4}.$$

Moreover,

[42] Recall that in the intended application for hardness amplification, the message length is $2^{O(T)}$ (i.e., the truth-table size of $f_x(\cdot)$), and n indicates the length of an additional fixed input $x \in \{0,1\}^n$.

- **Decoder complexity.** *Each C_i (as an oracle circuit) has size* $\mathrm{poly}(\log n, \log(1/\epsilon), 1/\delta)$ *and* $O(\log(1/\epsilon) + \mathrm{loglog}(\log(n)/\delta))$ $\mathbf{\Sigma}_2^w$*-oracle gates. The algorithm $\mathcal{A}$ generates each circuit in time polynomial in its size.*
- **Oracle complexity.** *The verifier for the* $\mathbf{\Sigma}_2^w$ *language runs in time* $\mathrm{poly}(\log n, \log(1/\epsilon), 1/\delta)$ *and uses* $O(1)$ *oracle queries to* w*.*
- **Encoding complexity.** *Each output symbol in the encoding of $\mathcal{C}$ is computable in time* $\mathrm{poly}(\log n, \log(1/\epsilon), 1/\delta)$*.*

Two shortcomings still remain: First, in the small-ϵ regime, the rate is quasi-polynomially small, rather than say, polynomially-small, or even almost-linearly small. Second, the decoder uses $\mathbf{\Sigma}_2$ oracle gates, whereas we might hope to only use **NP** gates. (Recall that *deterministic* decoding is provably impossible with $o(1/\epsilon)$ queries in our regime of parameters [DHP+26, Theorem 7.1].)

As explained in Section 2.4, the construction follows Dikstein *et al.* [DHP+26]: We first encode with the large-alphabet code $\mathcal{C}_{\mathrm{HDX}}$ of [DHP+26], and then encode each symbol with an inner code $\mathcal{C}_{\mathrm{in}}$. The inner code is a composition $\mathcal{C}_2 \circ \mathcal{C}_1$, where $\mathcal{C}_1$ is (approximately) locally uniquely decodable, and $\mathcal{C}_2$ is the direct product code of Impagliazzo *et al.* [IJK+10]. The *decoding*, however, follows the high-level idea in [DHP+26], but in order to get a logarithmic dependence on the agreement parameter, each decoder will use **NP** gates to "binary-search" through the input lists instead of querying entire lists. Decoding the concatenated code via a suitable consistency check will then require $\mathbf{\Sigma}_2$ gates. Additionally, in some places, we'll need to replace independent draws with randomness-efficient sampling. Details follow.

8.1 The code $\mathcal{C}_{\mathrm{HDX}}$

In this section we present the code from Theorem 1.13, which will serve as the outer code in the concatenated code from Theorem 1.14.[43]

The code is the HDX-based code of Dikstein *et al.* [DHP+26]. Our decoder uses $O(\log(1/\epsilon))$ oracle gates to **NP** to "binary-search" through the input lists instead of reading them in full (their size is *polynomial* in $1/\epsilon$). A bit more specifically, along a fixed HDX route, the **NP** witness supplies one list index at each coordinate (or, set on a route), and the verifier checks successive pairs using the intersection tests of [DHP+26]. We use standard search-to-decision to recover the lexicographically-first reachable and consistent final index. Following the analysis in [DHP+26], we show that from suitable starting advice, the function corresponding to this final index is sufficiently close to the target function with the required probability.

The code below is more general than the one stated in Theorem 1.13, since it is also *list-recoverable* rather than only list-decodable. Moreover, it also decodes the corrupt codeword has ϵ-fraction of coordinates on which it *approximately* agrees with the true

[43] The final decoder will not use Theorem 1.13 as a black box, but it will rely on the main notation and arguments introduced in this section.

codeword, tolerating $1/\mathrm{polylog}(n)$ error. (The special case of a list-decodable code is obtained from the statement below using $\ell = 1$ and $\delta' = 0$.)

Theorem 8.3 (Theorem 1.13, more general version). *For every $n, \ell \in \mathbb{N}$, alphabet Σ_0, and any $\epsilon, \delta \in (0,1)$, there exists an explicit code $\mathcal{C}_{\mathrm{HDX}} \colon \Sigma_0^n \to \Sigma^N$ with $N = \mathrm{poly}(1/\epsilon, 1/\delta, \ell) \cdot \widetilde{O}(n)$ and $\log|\Sigma| = \mathrm{poly}(1/\epsilon, 1/\delta, \ell, \log n) \cdot \log|\Sigma_0|$, such that the following holds.*

There exists a randomized algorithm $\mathcal{A}$ that generates randomized oracle circuits $C_1, \ldots, C_L$, for $L = O(\ell/\epsilon)$, such that for any input lists $\mathcal{L}_1, \ldots, \mathcal{L}_N \subseteq \Sigma$, each of size ℓ, and for every $f \in \Sigma_0^n$ that satisfies

$$\Pr_{r \in [N]} \left[\exists g \in \mathcal{L}_r : \Delta(g, \mathcal{C}_{\mathrm{HDX}}(f)_r) \leq \delta' \right] \geq \epsilon,$$

where $\delta' = \frac{\delta^2}{t^2}$ for $t = \mathrm{polylog}(n)$, it holds that

$$\Pr_{\mathcal{A}} \left[\exists i \in [L] : \Pr_{x \in [n], C_i} [C_i(x) = f(x)] \geq 1 - \delta \right] \geq \frac{3}{4}.$$

Moreover,

- **Decoder complexity:** *Each C_i (as an oracle circuit) has size $\mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|)$, and has $O(\log(\ell/\epsilon) + \log\log(t/\delta))$ $\mathbf{NP}^{\mathcal{L}}$ oracle gates and additional one oracle gate to $\mathcal{L}$. The algorithm $\mathcal{A}$ generates each C_i in time polynomial in its size.*
- **Oracle complexity:** *The verifier for the $\mathbf{NP}^{\mathcal{L}}$ language in the oracle runs in time $\mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|)$ and uses access to t elements of $\mathcal{L}$. Moreover, for each of the t queries, the decoder only reads $O\left(\frac{t}{\delta}\log\frac{t\ell}{\delta\epsilon}\right)$ Σ_0-symbols of the answer.*[44]
- **Encoding complexity.** *Each symbol in the encoding is computable in time $\mathrm{poly}(\log n, \ell, 1/\epsilon, 1/\delta) \cdot \log|\Sigma_0|$, after reading $\mathrm{poly}(\log n, \ell, 1/\epsilon, 1/\delta)$ locations of the message. Treating each symbol as a tuple of Σ_0-symbols, each such inner symbol can be encoded in $\mathrm{poly}(\log(n\ell|\Sigma_0|/(\epsilon\delta)))$ time, after reading one location of the message.*

Proof. Let (V, S, G) be the regular hypergraph system of [DHP+26, Proposition 4.3], set with $|V| = n$,[45] and we think of a message as a function $f \colon V \to \Sigma_0$. For the Guruswami–Indyk reduction [GI03], we take the same bipartite D-regular sampler $\Gamma = (S, R, E)$ that [DHP+26] uses, where $D = \mathrm{poly}(\ell, 1/\epsilon, 1/\delta, \log n)$, and we let $\hat{R}$ be the family of multisets

$$\hat{R} = \left\{ \biguplus_{i \in [D]} s_{\Gamma(r,i)} : r \in R \right\},$$

[44] The point of the "moreover" part is to get around the fact that symbols of the corrupt codeword have size $\log(|\Sigma|) \geq \mathrm{poly}(1/\epsilon)$, yet our decoder is not allowed to run in such time.

[45] To be more accurate, we can only guarantee that $|V|$ is chosen from some interval $n' \in [n, \mathrm{poly}(\ell, 1/\epsilon, 1/\delta, \log n) \cdot n]$. This can be addressed without a meaningful change in the parameters, and for simplicity of presentation we omit the handling of this nuance.

where $s_j \subseteq V$ denotes the j'th subset in S.

The code $\mathcal{C}_{\mathrm{HDX}}$ is the direct-product code $\mathcal{C}_{V,\hat{R}}$ that maps Σ_0^V to Σ^R, where $|R| = N$, and both $|\Sigma|$ and N are given above in the theorem's statement. To spell this out, for each $r \in R$, the rth symbol is the (concatenation of) the evaluations of f on every V-element in every set $s_{\Gamma(r,i)} \in S$; thus, the alphabet log-size is $D \cdot s = \mathrm{poly}(\ell, 1/\epsilon, 1/\delta, \log(n))$ where $s = \mathrm{poly}(\log n, 1/\epsilon)$ is the degree of each set in S in the hypergraph (V, S, G). We inherit the encoding time from [DHP+26], and specifically from the improved encoding construction that appears in [DHP+26, Section 6.7.7 (see comment at the end of the proof of Proposition 4.3)]. We will equip the hypergraph system (V, S, G) with the unit congestion routing scheme of [DHP+26, Proposition 4.3, Item (3)], whose length is $t = \mathrm{polylog}(n)$ as in our statement.

The first part of decoding: Guruswami–Indyk reduction. Consider input lists $\mathcal{L}_1, ..., \mathcal{L}_N$ to the decoder, an ϵ-fraction of which have an element that approximately agrees with the correct codeword on the corresponding symbol. To reduce to the high-agreement regime, the decoder applies the Guruswami–Indyk reduction exactly as in [DHP+26], and gets "simulated" inputs lists L_s for each $s \in S$.

Specifically, to obtain L_s we consider sampling edge-labels $d_1, \ldots, d_m \in [D]$ *once* for all $s \in S$, where $m = O(\log(t/\delta)/\epsilon)$. Moreover, we don't store $d_1, \ldots, d_m$ (since $m = \Omega(1/\epsilon)$, and that's too large for us), but rather sample the d_i-s in a k-wise independence manner for $k = O(\log(t/\delta))$, such that each d_i can be computed in time $\mathrm{poly}(\log D, \log m, k)$. Then, each L_s consists of all local evaluations of the corrupt codeword on s across the sampled neighbors r of s and the elements in the corresponding list $\mathcal{L}_r$. Specifically, denoting $\mathcal{L}_r = \{w_{r,1}, \ldots, w_{r,\ell}\}$, for each set $s \in S$ and edge-label $i \in [m]$ and list element $j \in [\ell]$ let

$$g_{s,(i,j)} = w_{\Gamma(s,d_i),j}|_s$$

and let

$$L_s = \left\{ g_{s,(i,j)} : i \in [m], j \in [\ell] \right\}.$$

We denote by $\ell' = m \cdot \ell$ the size of each L_s. The parameters of the construction are set such that the "local intersection graphs" are $(\delta/(t\ell'))^{\Omega(1)}$-samplers,[46] and Γ is an $((\epsilon\delta)/t)^{\Omega(1)}$-sampler. Assuming that f satisfies the hypothesis, the Guruswami–Indyk argument tells us that with high probability over the choice of $d_1, \ldots, d_m$, all but $O(\delta/t)$ fraction of $s \in S$ satisfy that there exists a $u \in [\ell']$ such that $\Delta(g_{s,u}, f|_s) \leq c\delta/t$ for a sufficiently small constant $c > 0$ (in fact, $\Omega(\epsilon/\ell)$ fraction of the u-s are such).

The latter statement is usually proved for fully independent choices of $d_1, ..., d_m$, and we argue that it holds also for k-wise independent choices. To see this, for each fixed s, note that the indicators that d_i satisfies the condition "there exists a $u \in [\ell']$ such that $\Delta(g_{s,u}, f|_s) \leq c\delta/t$" are k-wise independent; using standard tail bounds for k-wise independence, with probability at least $1 - O(\delta/t)$, there are $\Omega(\epsilon)$-fraction of

[46] Recall that this is the bipartite graph with the vertices in S on one side and pairwise-intersections $s \cap s'$ (for all $s' \in S$) on the other side.

the d_i's for which the test passes. Using Markov's inequality, with high probability over the seed for the k-wise independent generator, the same property still holds.

Verifying a good index-sequence. Recall that the decoding algorithm of [DHP+26, Section 3.1.2] uses a randomized *distance tester*, that gets as input two lists of functions, and tells us which functions in the first list are close to some function in the second list over their common domain.

For $s, s' \in S$ and $u, u' \in [\ell']$, we let $\mathcal{D}_{s,s'}(u, u')$ be the distance tester on $g_{s,u}$ and $g_{s',u'}$, that checks whether on $s \cap s'$, the distance between the two functions is $O(\delta/t)$. We use the same distance threshold, but crucially, we'll only test for distance between two individual functions (see [DHP+26, Claim F.1], using straightforward sampling) and the exact locations ℓ, ℓ' in the list will be supplied by the **NP**-witness. The distance tester will be the same as in [DHP+26], where we randomly subsample

$$d = O\left(\frac{t}{\delta} \log \frac{t\ell'}{\delta}\right)$$

coordinates per intersection, and we will reuse the same choice of coordinates in all oracle calls. The parameters are set such that we can guarantee the desired accuracy, with sufficiently high probability, simultaneously for every candidate pair on every route edge (recall that there are t such edges).

We define a non-deterministic algorithm that verifies that a partial index-sequence can be extended while satisfying pairwise-distances. Specifically, given a path $P = (s_1, \ldots, s_t)$ in G (which will later be the output of the routing scheme), a starting index $u_1 \in [\ell']$, and a prefix $p \in \{0,1\}^{\leq \log \ell'}$, we let $\mathsf{Reach}(P, u_1, p)$ be the following nondeterministic algorithm, that on witness $u_2, \ldots, u_t \in [\ell']$, does the following:

- For $i \in [t-1]$, if $\mathcal{D}_{s_i,s_{i+1}}(u_i, u_{i+1})$ rejects, reject.
- If the binary encoding of u_t does not begin with p, reject.
- Accept.

In words, for every u_1 and path P, the algorithm Reach accepts p iff there exist $u_2, \ldots, u_t \in [\ell']$ such every successive pair u_i, u_{i+1} passes the intersection test, and the binary representation of u_t starts with p. Note that each call to Reach makes at most t queries to $\mathcal{L}$ (and sub-samples d coordinates to read within each answer).

Remark 8.4. *Crucially,* Reach *does not have access to any fixed vertex v in the final set s_t. We insist on this because our later analysis needs g_{s_t,u_t} to be fixed before sampling a uniform $v \in s_t$ – in particular, conditioned on P, the selected u_t needs to be independent of v. The regularity of the hypergraph will allow us to replace $v \leftarrow V$ followed by $s_t \leftarrow \{s : v \in s\}$ by $s_t \leftarrow S$ followed by $v \leftarrow s_t$.*

Finding a good index-sequence. Equipped with the **NP** predicate Reach, we can return a canonical good u_t via a simple search-to-decision. Given P and u_1 as above, $\mathsf{FindSeq}(P, u_1)$ works as follows.

- If $\mathsf{Reach}(P, u_1, \emptyset) = 0$, declare failure.
- Set $p \leftarrow \emptyset$.
- For $i \in [\log \ell']$: If $\mathsf{Reach}(P, u_1, p \circ 0)$, set $p \leftarrow p \circ 0$, and otherwise set $p \leftarrow p \circ 1$.
- Return $u_t = p$.

Note that the procedure makes $\log \ell'$ calls to an $\mathbf{NP}^{\mathcal{L}}$ oracle.

The decoding algorithm. The algorithm that generates the decoding circuits draws uniformly at random $s_1 \in S$, and L starting indices $u_1^{(1)}, \ldots, u_1^{(L)} \in [\ell']$,[47] draws the randomness for the Guruswami–Indyk reduction, and outputs the i'th (randomized) decoding circuit $\mathsf{Dec}_{s_1, u_1^{(i)}}(v)$ that works as follows.

- Sample s_t uniformly at random out of all $s \in S$ that contain v.
- Compute $P = (s_1, \ldots, s_t) \leftarrow \mathcal{O}(s_1, s_t)$, where $\mathcal{O}$ is the (randomized) low congestion routing scheme of [DHP+26, Proposition 4.3]. Fix the randomness for the distance testers.
- Compute $u_t \leftarrow \mathsf{FindSeq}(P, u_1^{(i)})$. If the procedure failed, output an arbitrary fixed symbol.
- Query $\mathcal{L}$ to output $g_{s_t, u_t}(v)$.

We now argue that the decoding circuit's size and the runtime of the verifier (for the **NP** oracle) are as stated. Towards showing this, observe that the decoder and the verifier always access the lists via their indices, and never enumerate over their elements (and since $\ell' = m\ell$, each index is of size $\log(\ell') = O(\log(\ell/\epsilon) + \log\log(t/\delta))$). Also observe that the decoder and the verifier avoid enumerating over the Guruswami–Indyk samples; specifically, to compute $\Gamma(s, d_i)$ the decoder computes d_i from the bounded-independence seed in time $\mathrm{poly}(\log D, \log m, k)$. The hypergraph system and its intersections can likewise be computed in time $\mathrm{polylog}(1/\epsilon)$.[48]

Keeping the above in mind, for the decoding circuit, the dominant size factor is generating the intersection-test coordinates. The circuit generates $O(td)$ samples, and each takes size $\mathrm{poly}(\log n, \log(\ell/\epsilon), \log(1/\delta))$ using the efficient intersection circuits.

[47] Recall that the complexity guarantee we require is that for each $i \in [L]$ the ith circuit can be generated efficiently.

[48] The bound on the intersection circuit size does not readily follow from [DHP+26, Sections 6.7.5, 6.7.7], which only assert circuits of size $\mathrm{poly}(\log n, 1/\epsilon)$. However, inspecting their circuit construction, they use exhaustive search to eventually find a common representative in a subspace. Instead, we can use Gaussian elimination (followed by an efficient sampling from a certain subgroup) to do this in time $\mathrm{poly}(\kappa, \log q)$ (using their notation), which depends poly-logarithmically on $1/\epsilon$.

As for the verifier in the **NP** oracle, the witness supplies only t indices, using $O(t \log \ell')$ bits, and the verifier checks only the successive pairs specified by these indices. Each intersection test reads $d = \tilde{O}(t/\delta)$ sampled coordinates from each of its two functions. Thus, one verification queries $O(td)$ Σ_0-symbols overall (recall that we do not read entire Σ-symbols).

Correctness. The correctness follows the argument in [DHP+26, Theorem 3.4]. As explained above re the Guruswami–Indyk reduction (see also [DHP+26, Proof of Theorem 3.4 & Lemma 3.7]), with high probability over choice of s_1, an $\Omega(\epsilon/\ell)$-fraction of the elements in L_{s_1} are δ-close to f on s_1. Thus, with high probability over the coins of $\mathcal{A}$, at least one of the decoding circuits will satisfy this property.

In [DHP+26, Proof of Theorem 3.4] they defined a notion of a good path P, which is a path that satisfies the following for every edge (s_i, s_{i+1}) along the path and every $s \in \{s_i, s_{i+1}\}$: there is $g \in L_s$ that is δ-close to f, and every $g \in L_s$ satisfies $|\Delta_s(f,g) - \Delta_{s_i \cap s_{i+1}}(f,g)| \leq \delta$. They also showed that with high probability over choice of s_1 we have $\Pr_{s_t, \mathcal{O}}[P = \mathcal{O}(s_1, s_t) \text{ is good}] \geq 1 - \delta/2$.

The main claim established in the proof of [DHP+26, Theorem 3.4] is that whenever the path is good and the distance testers are correct and $\Delta(g_{s_1,u_1}, f|_{s_1}) \leq \delta$, for *every* index-sequence that satisfies all the pairwise distance testers tests, and every $g_{s_{i-1},u_{i-1}}, g_{s_i,u_i}$ along the index sequence, we have

$$\Delta(g_{s_i,u_i}, f|_{s_i}) \leq \Delta(g_{s_{i-1},u_{i-1}}, f|_{s_{i-1}}) + O(\delta/t) \; . \tag{8.1}$$

In particular, in this case, every function given to us by the last index is $O(\delta)$-close to $f|_{s_t}$ (and such an index-sequence exists).

Now, let us condition on the high-probability event that $\mathcal{A}$ chooses good randomness for the Guruswami–Indyk reduction, and s_1 such that $\Omega(\epsilon/\ell)$ of the elements in L_{s_1} are δ-close to f on s_1, and such that for $1 - O(\delta)$ of uniformly chosen s_t's, the chosen path P is good with high probability (over $\mathcal{O}$). Since $\Omega(\epsilon/\ell)$ of the elements in L_{s_1} are δ-close to f on s_1 and there are $L = O(\ell/\epsilon)$ choices of $u_1^{(i)}$, with high probability one of them is δ-close to f on s_1. Fixing such a decoding circuit, we want to show that it successfully decodes $1 - O(\delta)$ of the vertices.

As in [DHP+26], we use the regularity of the set system, and here we crucially rely on Remark 8.4. Specifically, recall that the randomness for the distance testers is fixed before calling FindSeq and Reach, and thus (except with low probability) $\mathsf{FindSeq}(P, u_1^{(i)})$ always finds the lexicographically first good index $u_t(P, u_1^{(i)}) \in s_t$, regardless of v. In particular, denoting the good decoding circuit by C_i and the event

that it chooses good randomness for the distance testers and for $\mathcal{O}$ by $\mathcal{E}$, we have

$$\begin{aligned}\Pr_v[C_i(v) = f(v)|\mathcal{E}] &= \Pr_{s_t, v\in s_t, C_i}[C_i(v) = f(v)|\mathcal{E}] \\ &\geq \Pr_{s_t}[\text{Eq. (8.1) holds}] \cdot \Pr_{v\in s_t}[C_i(v) = f(v)|\mathcal{E}, \text{Eq. (8.1) holds}] \\ &\geq (1-\delta/2) \cdot \Pr_{v\in s_t}[g_{s_t, u_t(P, u_1^{(i)})}(v) = f(v)|\mathcal{E}, \text{Eq. (8.1) holds}] \\ &\geq 1 - O(\delta)\end{aligned}$$

where the last inequality is due to Eq. (8.1). Hence, for $1 - O(\delta)$ of the vertices $v \in V$, the good decoding circuit succeeds with high probability $\Pr[\mathcal{E}]$. ∎

8.2 An inner code $\mathcal{C}_{\text{in}}$

We denote by m the message length for the inner code $\mathcal{C}_{\text{in}}$ that we will describe next, where we should think of $m = \log|\Sigma|$, under the notation of Theorem 8.3. We will use the code to τ-approximately recover each symbol of the outer code in Theorem 8.3, with $\tau = 1/\text{polylog}(n)$.

Lemma 8.5. *For every $m, \ell_{\text{in}} \in \mathbb{N}$ and any agreement parameter $\alpha > 0$ and $\tau = m^{-O(1)}$, there is a code $\mathcal{C}_{\text{in}}\colon \{0,1\}^m \to \overline{\Sigma}^{\binom{M}{s}}$, where $s = O(\log(\ell_{\text{in}}/\alpha))$ and $M = \text{poly}(1/\tau) \cdot m$ and $\overline{\Sigma} = \{0,1\}^s$, satisfying the following (approximate, local) list recovery guarantee.*

There exists a randomized algorithm $\mathcal{B}$ that generates randomized oracle circuits $D_1, \ldots, D_{L_{\text{in}}}$ for $L_{\text{in}} = O(\ell_{\text{in}}/\alpha)$, such that for any input lists $\mathcal{L} = \{\mathcal{L}_B\}_{B\in\binom{[M]}{s}}$, each of size ℓ_{in}, and for every $f \in \{0,1\}^m$ that satisfies

$$\Pr_{B\in\binom{[M]}{s}}[\mathcal{C}_{\text{in}}(f)_B \in \mathcal{L}_B] \geq \alpha,$$

it holds that

$$\Pr_{\mathcal{B}}\left[\exists i \in [L_{\text{in}}] : \Pr_{x\in[m], D_i}[D_i^{\mathcal{L}}(x) = f(x)] \geq 1 - \tau\right] \geq \frac{3}{4}.$$

Moreover,

- **Decoder complexity.** *Each D_i (as an oracle circuit) has size $\text{poly}(\log m, 1/\tau, \log(\ell_{\text{in}}/\alpha))$ and $Q_1 \cdot \log(\ell_{\text{in}}/\alpha)$ $\mathbf{NP}^{\mathcal{L}}$ oracle gates and Q_1 additional oracle gates to $\mathcal{L}$, where $Q_1 = \text{polylog}(\log(m)/\tau)$.*

  The algorithm $\mathcal{B}$ uses a single random seed of length $\text{poly}(\log m, 1/\tau, \log(\ell_{\text{in}}/\alpha))$ to generate all circuits. Given that seed, and the corresponding index $i \in [L_{\text{in}}]$, $\mathcal{B}$ generates the i'th circuit D_i in time polynomial in that bound and in $1/\tau$.

- **Oracle complexity.** *The verifier for $\mathbf{NP}^{\mathcal{L}}$ language in each D_i's oracle runs in time $\text{poly}(\log m, \log(1/\tau), \log(\ell_{\text{in}}/\alpha))$ and uses two queries to $\mathcal{L}$, reading s bits overall.*

- **Encoding complexity.** *Each encoded symbol is computable in time $s \cdot \text{poly}(\log m, 1/\tau)$, using at most $s \cdot \text{poly}(1/\tau)$ queries to the message.*

The code $\mathcal{C}_{\text{in}}$ will be a composition of two codes, $\mathcal{C}_1 \colon \{0,1\}^m \to \{0,1\}^M$ and $\mathcal{C}_2 \colon \{0,1\}^M \to (\{0,1\}^s)^{\binom{M}{s}}$. Starting with a string that has a small agreement α with some codeword (in our intended application $\alpha \approx \delta$), the decoder for $\mathcal{C}_2$ will (list-)recover a string that has 0.99 agreement with a corresponding codeword in $\mathrm{Im}(\mathcal{C}_1)$, and the decoder for $\mathcal{C}_1$ will recover a string that has agreement $1-\tau$.

The code $\mathcal{C}_1$. $\mathcal{C}_1$ will be the (locally) uniquely decodable code of [DHP+26, Proposition A.7], with the following guarantee. For any $z \in \{0,1\}^M$ that satisfies $\Delta(z, \mathcal{C}_1(x)) \leq 0.01$, using

$$Q_1 = \mathrm{polylog}(1/\tau, \log m)$$

queries to z, the decoding algorithm $\mathcal{A}_1$ satisfies $\Pr_{i \in [m], \mathcal{A}_1}[\mathcal{A}_1^z(i) = x_i] \geq 1-\tau$, where we assume that $\tau \geq m^{-O(1)}$. Under this choice of parameters,

$$M = \mathrm{poly}(1/\tau) \cdot m.$$

The encoding time, for a specific bit, is $\mathrm{poly}(\log m, 1/\tau)$, and we only need to query $\mathrm{poly}(1/\tau)$ bits of the message. The decoder (oracle) circuit is logspace-uniform, and has size which is polynomial in $\log m$ and $1/\tau$.

The code $\mathcal{C}_2$. For $s = O(\log(\ell_{\text{in}}/\alpha))$ as in the statement, let

$$\mathcal{C}_2 \colon \{0,1\}^M \to \overline{\Sigma}^{\binom{M}{s}},$$

for $\overline{\Sigma} = \{0,1\}^s$, be the naive direct-product code (or, "complete-complex code") from [DHP+26, Theorem A.10], following [IJK+10], set to work for agreement $\alpha > 0$.

Let us first recall the decodability properties of $\mathcal{C}_{\text{in}}$. Combining [DHP+26, Proposition A.7 & Theorem A.10], we know that given an input list size ℓ_{in}, the code $\mathcal{C}_{\text{in}}$ is list recoverable in the following sense. There exists a randomized algorithm that outputs a circuit C with oracle to input lists $\mathcal{L}_{\text{in}}$, such that for any $f \in \{0,1\}^m$ such that $\mathcal{C}_{\text{in}}(f)$ has α-agreement with $\mathcal{L}_{\text{in}}$, with high probability over the generating algorithm, there exists $i \in [L_{\text{in}}]$, where $L_{\text{in}} = O(\ell_{\text{in}}/\alpha)$, such that

$$\Pr_{x \in [m], C}[C^{\mathcal{L}_{\text{in}}}(x)_i = f(x)] \geq 1-\tau.$$

In the terminology of [DHP+26], the code is τ-approximate, ℓ_{in}-input $(\alpha, L_{\text{in}}, Q)$ *single-circuit* aLLRC. However, for us, here too, the query complexity Q is prohibitively large, so we use nondeterministic decoding.

Non-deterministic local decoding of $\mathcal{C}_{\text{in}}$: first implementation. We start by recalling the simple [IJK+10] local decoding algorithm for $\mathcal{C}_2$, adapted for list recovery. Similarly to the HDX case, we will use binary search to find a canonical consistent continuation.

The algorithm $\mathcal{B}$ chooses $R = O(1/\alpha)$ starting pairs $B_{a,0} \in \binom{[M]}{s}$ and $A_a \subseteq B_{a,0}$ of size $s/2$ uniformly at random (indexed by $a \in [R]$). Consider the local decoding circuit Dec, parameterized by $a \in [R]$ and $j_0 \in [\ell_{\text{in}}]$, that on input x works as follows.

- If $x \in A_a$, return $\mathcal{L}^{\text{in}}_{B_{a,0}}[j_0](x)$.
- Draw $B_{a,1}, \ldots, B_{a,q} \in \binom{[M]}{s}$ uniformly at random, conditioned on each $B_{a,i}$ containing $A_a \cup \{x\}$, where $q = O(1/\alpha)$.
- Set $p \leftarrow \emptyset$. For $i \in [\log(q \cdot \ell_{\text{in}})]$: If $\mathsf{Cons}(B_{a,0}, A_a, B_{a,1}, \ldots, B_{a,q}, j_0, p \circ 0)$ defined below, with oracle to $\mathcal{L}_{\text{in}}$, returns 1, set $p \leftarrow p \circ 0$, and otherwise set $p \leftarrow p \circ 1$.
- Letting $(h, j) = p$, return $\mathcal{L}^{\text{in}}_{B_{a,h}}[j](x)$.

The predicate Cons guesses a witness $(h, j) \in [q] \times [\ell_{\text{in}}]$ that extends p, queries $\mathcal{L}^{\text{in}}_{B_{a,0}}[j_0]$ and $\mathcal{L}^{\text{in}}_{B_{a,h}}[j]$ on A_a, and accepts iff they both agree. Note that Cons returns the first consistent set $B_{a,h}$ that one would encounter if they iterated through the sample $B_{a,1}, B_{a,2}, \ldots, B_{a,q}$ in-order, and that each $B = B_{a,h}$ is uniquely associated with the lexicographically first $j = j(B) \in [\ell_{\text{in}}]$ such that $\mathcal{L}^{\text{in}}_{B_{a,0}}[j_0]$ and $\mathcal{L}^{\text{in}}_{B_{a,h}}[j]$ agree on A_a. In particular, the selection of $B_{a,h}$ is identical to [IJK+10], and the choice of $j = j(B)$ is independent of x. Hence, the correctness follows from [IJK+10], as extended in [DHP+26].

Note that the algorithm that generates the decoding circuits fixes the randomness used for drawing the $B_{a,i}$'s, and with probability $\Omega(\alpha)$ over the external randomness and a starting pair, some j_0 defines a function within distance 0.01 to $\mathcal{C}_1(f)$. Thus, R trials suffice to amplify the success probability to $3/4$.

Non-deterministic local decoding of $\mathcal{C}_{\text{in}}$: actual implementation. The implementation above of the decoding circuit and the $\mathbf{NP}^{\mathcal{L}_{\text{in}}}$ verifier requires size that depends on $1/\alpha$, due to our bound on q. Moreover, using full independence to generate all R starting pairs will be costly for us when we consider the final concatenated code.

To remedy that, and obtain a polylogarithmic dependence on $1/\alpha$, we will draw both the starting pairs and the $B_{a,i}$'s using an pairwise-independence generator.

For the starting pairs, pairwise independence trials suffice to get constant success probability. Specifically, for reasons that will become clear below, what we want is to "hit" $(B_{a,0}, A_a)$ such that there exists $j_0 \in [\ell_{\text{in}}]$ for which: (1) the starting advice $\mathcal{L}^{\text{in}}_{B_{a,0}}[j_0]|_{A_a}$ is correct; (2) a uniform continuation $B \supseteq A_a \cup \{x\}$ (which may be of size $|B| < s$) has a consistent continuations (up to a set of full size s) with sufficiently high probability, for sufficiently many $x \in [M] \setminus A_a$'s; and (3) For a typical $x \in [M] \setminus A_a$, the probability that the first agreeing entry is wrong, conditioned on the sampled continuation having an agreeing entry, is small. For uniformly random choices of $B_{a,0}$ and $A_a \subseteq B_{a,0}$, the analysis in [IJK+10] (adapted to list recovery) shows that the event happens with probability $\Omega(\alpha)$, and thus this event also happens with probability $\Omega(\alpha)$ in the pairwise-independent choice.

Now, given a successful starting pair, we argue that pairwise-independence among the $B_{a,i}$'s suffices to carry through the rest of the analysis. Specifically, we sample $B \supseteq A_a \cup \{x\}$ and the event we want to hit is simply that B's list contains an entry agreeing with our advice on A_a – this is condition (2) above, instantiated for the case of a full continuation (i.e., for $|B| = s$, in which case the required probability in (2) is

$\Omega(\alpha)$). Given this guarantee, to argue that the decoder outputs the correct answer on most x's, we use condition (3) above.

By a standard construction of bounded independence, the two seeds have length

$$O\left(s \log M + \log(1/\alpha)\right) = \mathrm{poly}(\log(\ell_{\mathrm{in}}/\alpha), \log m, \log(1/\tau))$$

and each set, given the seed, is computed in size polynomial in the bound above.

The algorithm $\mathcal{B}$ can thus generate the ith decoding circuit (from the bounded independence seed) in time polynomial in the bound above. Also, each circuit will not pass all of the $B_{a,i}$'s to the oracle $\mathbf{NP}^{\mathcal{L}_{\mathrm{in}}}$ verifier. Instead, for a given prefix p, the input to the verifier is $B_{a,0}, A_a$, a bounded independence seed for choosing the $B_{a,i}$'s, and $p \circ 0$. The verifier guesses a witness $(h, j) \in [q] \times [\ell_{\mathrm{in}}]$ that extends p, computes (only) $B_{a,h}$ from the bounded independence seed, and checks agreement on A_a.

Thus, neither $\mathcal{B}$ nor the decoding circuits enumerates over q or R, and the asserted bounds on their runtime and size follow. The total number of queries the decoding circuit makes to the $\mathbf{NP}^{\mathcal{L}_{\mathrm{in}}}$ oracle is $O(\log(\ell_{\mathrm{in}}/\alpha))$, and it also makes an additional list query (for the output). The oracle verifier uses two queries to the input lists.

Once we have the decoding algorithm for $\mathcal{C}_2$, a decoding algorithm for $\mathcal{C}_{\mathrm{in}}$ readily follows (we run the decoder $\mathcal{A}_1$ for $\mathcal{C}_1$ with oracle access to each fixed $\mathcal{C}_2$ candidate).

8.3 The concatenated code

Recall that $\mathcal{C}_{\mathrm{HDX}} \colon \Sigma_0^n \to \Sigma^N$ and $\mathcal{C}_{\mathrm{in}} \colon \{0,1\}^m \to \overline{\Sigma}^{\binom{M}{s}}$, where $\overline{\Sigma} = \{0,1\}^s$. Our final code is the concatenated code $\mathcal{C} \colon \Sigma_0^n \to \overline{\Sigma}^{N' = N \cdot \binom{M}{s}}$. The following statement is a more general version of Theorem 1.14, which supports list-recovery (rather than only list-decoding).

Theorem 8.6 (the general case of Corollary 8.2 that allows list-recovery)**.** *For any $n, \ell \in \mathbb{N}$, alphabet Σ_0, and $\epsilon, \delta \in (0,1)$, the code $\mathcal{C} \colon \Sigma_0^n \to \overline{\Sigma}^{N'}$ above is δ-approximate ℓ-input (ϵ, L) locally list recoverable with $\log|\overline{\Sigma}| = O(\log(\ell/\epsilon))$,*

$$L = O\left(\frac{\ell}{\epsilon} \cdot \log^2 \frac{1}{\epsilon}\right),$$

and

$$N' = \left(\frac{\ell}{\epsilon\delta} \cdot \log(n) \log|\Sigma_0|\right)^{O(\log(\ell/\epsilon))} \cdot n.$$

In particular, there exists a randomized algorithm $\mathcal{A}$ that gnereates randomized oracle circuits $C_1, \ldots, C_L$, such that for any input lists $\{\mathcal{L}_z\}_{z \in [N']}$, each $\mathcal{L}_z \subseteq \overline{\Sigma}$ of size at most ℓ, and every $f \in \Sigma_0^n$ that satisfies

$$\Pr_{z \in [N']}\left[\mathcal{C}(f)_z \in \mathcal{L}_z\right] \geq \epsilon,$$

it holds that

$$\Pr_{\mathcal{A}}\left[\exists i \in [L] : \Pr_{x \in [n], C_i}\left[C_i^{\mathcal{L}}(x) = f(x)\right] \geq 1 - \delta\right] \geq \frac{3}{4}.$$

Moreover,

- **Decoder complexity.** *Each C_i (as an oracle circuit) has size* $\mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|)$, *and* $O(\log(\ell/\epsilon) + \log|\Sigma_0| + \log\log((\log n)/\delta))$ $\mathbf{\Sigma}_2^{\mathcal{L}}$ *oracle gates. The algorithm $\mathcal{A}$ generates each circuit in time polynomial in its size.*
- **Oracle complexity.** *The verifier for the* $\mathbf{\Sigma}_2^{\mathcal{L}}$ *language runs in time* $\mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|)$ *and uses $O(1)$ queries to $\mathcal{L}$ (and hence $O(\log|\overline{\Sigma}|)$ bits).*
- **Encoding complexity.** *Each output symbol in the encoding of $\mathcal{C}$ is computable in time* $\mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|)$.

Proof. At a high level, we follow the concatenation argument in [DHP+26, Proposition A.4] using inner single-circuit decoders, and the "bucketing" argument.

The concatenated code. For the outer code, we use $\mathcal{C}_{\mathrm{HDX}}$ with message length n, alphabet Σ_0, designated approximation error δ, input-list bound $\ell_{\mathrm{HDX}} = O(\ell/\epsilon)$, and agreement $O(\epsilon/T)$. Set N, Σ, and δ' accordingly, as in Theorem 8.3, and now we instantiate $\mathcal{C}_{\mathrm{in}}$ with message length $m = \log|\Sigma|$, input-list bound ℓ, agreement $\epsilon/2$, and approximation error $\tau = O(\delta'/\log|\Sigma_0|)$. Let $s = O(\log(\ell/\epsilon))$ and $M = m \cdot \mathrm{poly}(1/\tau)$ be as in Lemma 8.5. The bound on N' follows by plugging-in the parameters, recalling that $N' = N \cdot \binom{M}{s}$.

Encoding complexity. For the encoding queries, recall that each symbol of $\mathcal{C}_{\mathrm{in}}$ depends on $s \cdot \mathrm{poly}(1/\tau)$ bits in (the binary encoding of) a Σ-symbol in the output of $\mathcal{C}_{\mathrm{HDX}}$. In turn, each inner Σ_0-symbol in a Σ-symbol depends on one input symbol. Thus, each output symbol depends on

$$s \cdot \mathrm{poly}(1/\tau) = \mathrm{poly}(\log n, \log(|\Sigma_0|), 1/\delta, \log(\ell/\epsilon))$$

input symbols.

For the encoding time of each output symbol, again combining Theorem 8.3 and Lemma 8.5 we get

$$s \cdot \mathrm{poly}(\log m, 1/\tau) + s \cdot \mathrm{poly}(1/\tau) \cdot \mathrm{poly}(\log n, \log(\ell/\epsilon), \log(1/\delta), \log|\Sigma_0) = \mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|).$$

Decoding. We closely follow the high-level construction and analysis of [DHP+26, Claim A.5], and in particular the bucketing argument. Letting $T = O(\log(1/\epsilon))$ be the number of buckets, we run the decoder described next for every bucket $k \in [T]$ (and keep all the output circuits).[49]

[49] We note that in bucket k, we run the decoder with a smaller list bound of $O(\ell/(2^k\epsilon))$ and larger agreement $O(2^{-k}/T)$, but we skip those details, as they are identical to [DHP+26].

The algorithm $\mathcal{A}$ generating the decoding circuits first chooses randomness for $\mathcal{B}$ (i.e., seeds for the circuit-generating algorithm $\mathcal{B}$ for $\mathcal{C}_{\text{in}}$), where a single random choice will be used across all blocks, and the randomness it needs for the outer decoding, specifically for the Guruswami–Indyk reduction. This implicitly generates the inner candidates guaranteed to us by Lemma 8.5 for each outer block, and defines one deterministic circuit on $[m]$, for each block and each candidate index. Following the analysis in [DHP+26], for each good block (in the sense of the bucketing argument) the list provided by the inner decoder contains a "δ'-good" function with high enough probability, and hence with constant probability, the lists in a constant fraction of the good blocks contain a δ'-good candidate. By a constant number of independent repetitions (and absorbing the constant loss above into the bucketing parameters), with probability at least $3/4$ the circuit-generation procedure succeeds. Each bucket contributes $O(\ell T/\epsilon)$ decoding circuits, so the overall number of decoding circuits is

$$L = O(\ell T^2/\epsilon) = O((\ell/\epsilon)\log^2(1/\epsilon)) \ .$$

The upper-bound on the size of each decoding circuit follows directly from Theorem 8.3 and Lemma 8.5. As for the complexity of the oracle used by the decoding circuits, recall that the decoder of $\mathcal{C}_{\text{HDX}}$ uses $\mathbf{NP}^{\mathcal{L}}$ gates (to test "reachability" along a fixed route, i.e. find the lex-first last index u_t to which there is a pairwise-consistent sequence of indices along the route). In the decoder for the concatenated code, we provide access to $\mathcal{L}$ by simulating the (fixed) inner candidates from Lemma 8.5. The combination yields a $\mathbf{\Sigma}_2$ verifier, and we present below a more careful implementation of a $\mathbf{\Sigma}_2$ verifier that will yield the stated parameters.

To bound the number of queries to the oracle, recall from the proof of Theorem 8.3 that the decoding circuit recovers u_t bit-by-bit using the $\mathbf{\Sigma}_2^{\mathcal{L}}$ oracle, starting with $p = \emptyset$, and at each step check whether there exists a consistent outer index-sequence whose terminal index extends $p \circ 0$. The query supplies P, $u_1^{(i)}$, and p, together with the fixed seeds and distance-tester samples. It then recovers its output symbol, and this uses altogether $O(\log \ell' + \log|\Sigma_0|)$ oracle gates.

A more fine-grained $\mathbf{\Sigma}_2$-verifier. The verifier's design will rely on details from the proofs of Theorem 8.3 and Lemma 8.5, and we use the terminology and notation from these proofs. Recall that the oracle query's input, as in the proof of Theorem 8.3, is a prefix p, a path $P = (s_1, s_2, ..., s_t)$ and a start-index $u_1 \in [\ell']$.[50] Also recall that the inner code is the composition of a uniquely decodable code $\mathcal{C}_1$ with the direct-product code $\mathcal{C}_2$.

- The *existential* phase guesses the outer indices $u_2, ..., u_t$ for the decoder of $\mathcal{C}_{\text{HDX}}$, and for each u_i it guesses the inner indices $(h^{(i)}, j^{(i)})$ for the decoder of $\mathcal{C}_2$, and the (supposed) output bits of $\mathcal{C}_2$ needed to simulate the local decoder of $\mathcal{C}_1$. (Recall that $\mathcal{C}_2$ outputs a symbol with $\log(|\Sigma|) \geq \text{poly}(1/\epsilon)$ bits, which the verifier

[50] As in that proof, for simplicity we omit the seeds for the pairwise-independence generators and the randomness for the distance testers, which have already been fixed at that point.

cannot afford to guess and store, but the decoder of $\mathcal{C}_1$ is local and only reads $\mathrm{polylog}(\log n, 1/\epsilon)$ bits, and the verifier guesses and stores only this many bits.) The verifier then simulates $\mathcal{A}_1$ to obtain the relevant symbols for each outer index, and performs the consistency tests (i.e., runs the distance testers). It also checks that u_t extends the prefix p.

(Note that for each outer index u_i the verifier computes $a_i, j_{0,i}$ and $(B_{a_i,0}, A_{a_i})$. Thus, although the initial u_1 is fixed in a particular decoding circuit, we need an efficient way to generate the starting pairs, and that is where our bounded independence generator in the proof of Lemma 8.5 helps.)

- The *universal* phase quantifies over all $i \in [t]$ and all output bits of $\mathcal{C}_2$, and checks that the output of $\mathcal{C}_2$ on $(h^{(i)}, j^{(i)})$ is consistent with the advice $A_{a_i,j_{0,i}}$ (note that after quantifying over all $\log(|\Sigma|)$ output bits, this check ensures that $\mathcal{L}^{\mathsf{in}}_{B_{a_i,h}}[j]_{A_{a_i}} = \mathcal{L}^{\mathsf{in}}_{B_{a_i,0}}[j_{0,i}]_{A_{a_i}}$). In addition, within this universal quantification, if the output bit of $\mathcal{C}_2$ is one of those needed for local decoding by $\mathcal{C}_1$, it also checks that the existential guess of this bit was correct.[51] Note that these verifications use two queries to $\mathcal{L}$.

  For each $i \in [t]$, the universal witness also selects $(h', j') < (h, j)$ and checks that it is inconsistent with the starting advice (using another two queries to $\mathcal{L}$).[52]

After recovering the first reachable final index u_t, the verifier evaluates its selected inner candidate, with one oracle gate per output bit.

For the verifier runtime, note that the number of $\mathcal{C}_2$-evaluations in the same as the number of queries to $\mathcal{L}$ in [DHP+26], which is $O(Q_1 \cdot Q_{\mathrm{out}})$, where $Q_1 = \mathrm{polylog}((\log m)/\tau)$ is as given above, and $Q_{\mathrm{out}} = O(td \log |\Sigma_0|)$, where d is the number of samples in the distance tester. Accounting for the outer t witnesses encoded using $O(t \log \ell')$ bits, and the fact that we need $O(\log(q\ell))$ bits to store each $\mathcal{C}_2$-evaluation, the witness length is therefore $O(t \log \ell' + Q_{\mathrm{out}} Q_1 \log(q\ell))$, and the verifier makes a constant number of queries. Working out the parameters, and recalling that $\ell' = O((\ell T/\epsilon)\log(t/\delta))$, the witness length amounts to $\mathrm{poly}(\log n, 1/\delta, \log(\ell/\epsilon), \log|\Sigma_0|)$. We also need to account for mundane intermediate computations in the outer and inner decoding (e.g., storing indices), simulating $\mathcal{A}_1$, consistency checks, and bounded-independence evaluation. Overall, following Lemma 8.5 and Theorem 8.3, the runtime is bounded by

$$\mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|) + Q_{\mathrm{out}} \cdot \mathrm{poly}(\log m, 1/\tau) +$$
$$Q_{\mathrm{out}} Q_1 \cdot \mathrm{poly}(\log N, \log M, \log(1/\epsilon), \log \ell, s) = \mathrm{poly}(\log n, \log(\ell/\epsilon), 1/\delta, \log|\Sigma_0|).$$

■

[51] The purpose of performing this last check inside a universal quantifier is to save on queries to $\mathcal{L}$.

[52] The selected pair is checked regardless of whether an earlier pair exists. If the existential witness instead indicated that no consistent pair exists, the verifier checks that the claimed output is the chosen fixed default bit and that (h', j') is inconsistent.

9 Open Problems

We curate a small list of the most prominent open problems arising from our work. The first one asks to improve the parameters of the codes in Section 1.4, which will yield better hardness amplification results for polynomial-time algorithms:

Open Problem 1. *Improve the parameters of the codes in Theorems 1.13 and 1.14. Ideally, construct a locally encodable code with polynomial rate and a doubly-local list-decoder from agreement $\delta(n) = 2^{-\Omega(n)}$ that runs in time $O(\log(n/\delta))$ and uses an $\mathcal{NP}$-oracle.*

The next problem asks to make further progress in the design of targeted PRGs and targeted samplers, by improving the underlying technical construction. Specifically, in [CT21a] the construction relied on non-batch-computability, and in Theorems 1.9 and 6.3 the construction relies on hardness of small circuits whose gates compute "big" low-degree arithmetic circuits. Can this be improved further, to deduce superfast derandomization from more relaxed hardness assumptions?

Open Problem 2. *Construct overhead-free targeted PRGs or targeted samplers from hardness for a broader class of functions, compared to circuits with low-degree gates as in Theorem 1.9.*

Our study of targeted samplers is based on hardness assumptions that are more relaxed than those required for targeted PRGs, and that (with appropriate hardness amplification using doubly-local decoders) can be deduced from more standard assumptions. However, we don't know whether the assumptions are necessary!

Open Problem 3. *Discover necessary hardness assumptions for the existence of targeted samplers with $D \ll \log(1/\delta)$. Then, ideally, construct targeted samplers from hardness assumptions that are also necessary for their existence (proving an "if and only if").*

Next, our quantified derandomization algorithm in Theorem 1.6 relies on hardness for uniform algorithms, but the hardness is a time-space tradeoff rather than standard hardness. We can alternatively deduce such derandomization from circuit lower bounds (see Section 4.3.1), which are standard but need to hold for non-uniform circuits. Can we have the best of both worlds?

Open Problem 4. *Deduce quantified derandomization as in the conclusion of Theorem 1.6 from standard hardness for uniform algorithms (rather than a time-space tradeoff), or as a first step, from standard hardness for probabilistic non-deterministic algorithms.*

The last main open problem addresses the most important current question in the study of superfast derandomization: is overhead-free derandomization possible? In other words, can we beat the time overhead of $T \mapsto O(n \cdot T)$ given by the non-uniform derandomization of Adleman [Adl78] (and matched conditionally in [CT21b])?

Open Problem 5. *Find evidence indicating what is the minimal time overhead for derandomization of probabilistic algorithms. Two concrete challenges are to find evidence for the possibility that $prBPTIME[T] \subseteq prDTIME[O(T)]$, or to find evidence against the possibility that $prBPTIME[T] \subseteq prDTIME[n^{1-\Omega(1)} \cdot T]$.*

In Appendix A we discuss the currently known evidence against the possibility that $prBPTIME[T] \subseteq prDTIME[n^{1-\Omega(1)} \cdot T]$.

Acknowledgements

We are grateful to Max Hopkins for explaining parts of [DHP+26] to us and for useful conversations about doubly-local decoders. We are grateful to Rahul Santhanam for telling us about the relevant prior work of Drucker [Dru13]. We thank Lijie Chen, Ted Pyne, Salil Vadhan, Valentine Kabanets, and David Zuckerman for useful conversations in early stages of this project.

Dean Doron is supported in part by Israel Science Foundation grant #857/25. Roei Tell is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN- 2024-04490. M. H. Ebtehaj is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2019-04804.

Part of this work was done while Heda Chen, Dean Doron, and Roei Tell were participating in the program "Pseudorandomness and High-Dimensional Expanders" at the Simons Institute for the Theory of Computing. Part of this work was done while Roei Tell was participating in the workshop "Frontiers in complexity lower bounds" at the Isaac Newton Institute for Mathematical Sciences, Cambridge.

References

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

[ACR+99] Alexander E. Andreev, Andrea E. F. Clementi, José D. P. Rolim, and Luca Trevisan. "Weak Random Sources, Hitting Sets, and BPP Simulations". In: *SIAM J. Comput.* 28.6 (1999), pp. 2103–2116. DOI: 10.1137/S0097539797325636. URL: https://doi.org/10.1137/S0097539797325636.

[Adl78] Leonard Adleman. "Two theorems on random polynomial time". In: *Proc. 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1978, pp. 75–83.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P". In: *Annals of Mathematics. Second Series* 160.2 (2004), pp. 781–793.

[AT19] Nir Aviv and Amnon Ta-Shma. "On the entropy loss and gap of condensers". In: *ACM Transactions on Computation Theory* 11.3 (2019), Art. 15, 14.

[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.

[BCT25] Marshall Ball, Lijie Chen, and Roei Tell. "Towards Free Lunch Derandomization from Necessary Assumptions (and OWFs)". In: *Proc. 40th Annual IEEE Conference on Computational Complexity (CCC)*. 2025.

[BDT19] Avraham Ben-Aroya, Dean Doron, and Amnon Ta-Shma. "An Efficient Reduction from Two-Source to Nonmalleable Extractors: Achieving Near-Logarithmic Min-Entropy". In: *SIAM Journal on Computing* 51.2 (Jan. 2019), STOC17–31–STOC17–49.

[BDT20] Avraham Ben-Aroya, Dean Doron, and Amnon Ta-Shma. "Near-optimal erasure list-decodable codes". In: *35th Computational Complexity Conference (CCC 2020)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2020, pp. 1–1.

[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.

[BGK06] Jean Bourgain, Alexey A Glibichuk, and Sergei Vladimirovich Konyagin. "Estimates for the number of sums and products and for exponential sums in fields of prime order". In: *Journal of the London Mathematical Society* 73.2 (2006), pp. 380–398.

[BIW06] Boaz Barak, Russell Impagliazzo, and Avi Wigderson. "Extracting Randomness Using Few Independent Sources". In: *SIAM J. Comput.* 36.4 (2006), pp. 1095–1118. DOI: 10.1137/S0097539705447141. URL: https://doi.org/10.1137/S0097539705447141.

[BKS+10] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. "Simulating independence: New constructions of condensers, ramsey graphs, dispersers, and extractors". In: *J. ACM* 57.4 (2010), 20:1–20:52. DOI: 10.1145/1734213.1734214. URL: https://doi.org/10.1145/1734213.1734214.

[BKT04] Jean Bourgain, Nets Katz, and Terence Tao. "A sum-product estimate in finite fields, and applications". In: *Geometric & Functional Analysis GAFA* 14.1 (2004), pp. 27–57.

[BM88] László Babai and Shlomo Moran. "Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes". In: *Journal of Computer and System Sciences* 36.2 (1988), pp. 254–276.

[CH20] Kuan Cheng and William M. Hoza. "Hitting Sets Give Two-Sided Derandomization of Small Space". In: *35th Computational Complexity Conference, CCC 2020, Saarbrücken, Germany (Virtual Conference), July 28-31, 2020*. Ed. by Shubhangi Saraf. Vol. 169. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 10:1–10:25. DOI: 10.4230/LIPICS.CCC.2020.10. URL: https://doi.org/10.4230/LIPIcs.CCC.2020.10.

[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.

[DHP+26] Yotam Dikstein, Max Hopkins, Toniann Pitassi, and Russell Impagliazzo. "High rate efficient local list decoding from HDX". In: *Proc. 58th Annual ACM Symposium on Theory of Computing (STOC)*. 2026, pp. 1789–1799.

[DKS+13] Zeev Dvir, Swastik Kopparty, Shubhangi Saraf, and Madhu Sudan. "Extensions to the Method of Multiplicities, with Applications to Kakeya Sets and Mergers". In: *SIAM J. Comput.* 42.6 (2013), pp. 2305–2328. DOI: 10.1137/100783704. URL: https://doi.org/10.1137/100783704.

[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.

[DMO+26] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. "Pseudorandomness Beating the Hybrid Argument for Insensitive Algorithms". In: *Electronic Colloquium on Computational Complexity: ECCC* 33 (2026), p. 082.

[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.

[DPW14] Yevgeniy Dodis, Krzysztof Pietrzak, and Daniel Wichs. "Key derivation without entropy waste". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2014, pp. 93–110.

[Dru13] Andrew Drucker. "Nondeterministic direct product reductions and the success probability of SAT solvers". In: *Proc. 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2013, pp. 736–745.

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

[DW08] Zeev Dvir and Avi Wigderson. "Kakeya Sets, New Mergers and Old Extractors". In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society, 2008, pp. 625–633. DOI: 10.1109/FOCS.2008.23. URL: https://doi.org/10.1109/FOCS.2008.23.

[GI03] Venkatesan Guruswami and Piotr Indyk. "Linear time encodable and list decodable codes". In: *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*. 2003, pp. 126–135.

[GKL+22] Halley Goldberg, Valentine Kabanets, Zhenjian Lu, and Igor C. Oliveira. "Probabilistic Kolmogorov Complexity with Applications to Average-Case Complexity". In: *37th Computational Complexity Conference, CCC 2022, Philadelphia, PA, USA, July 20-23, 2022*. Ed. by Shachar Lovett. Vol. 234. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 16:1–16:60. DOI: 10.4230/LIPICS.CCC.2022.16. URL: https://doi.org/10.4230/LIPIcs.CCC.2022.16.

[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.

[GLR+91] Peter Gemmell, Richard J. Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. "Self-Testing/Correcting for Polynomials and for Approximate Functions". In: *Proc. 23rd Annual ACM Symposium on Theory of Computing (STOC)*. 1991, pp. 32–42.

[Gol11a] Oded Goldreich. "A sample of samplers: a computational perspective on sampling". In: *Studies in complexity and cryptography*. Vol. 6650. Lecture Notes in Computer Science. Springer, Heidelberg, 2011, pp. 302–332.

[Gol11b] 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.

[Gol11c] Oded Goldreich. "Two Comments on Targeted Canonical Derandomizers". In: *Electronic Colloquium on Computational Complexity: ECCC* 18 (2011), p. 47.

[Gol24] Oded Goldreich. "On defining PPT-search problems". In: *Electronic Colloquium on Computational Complexity: ECCC* 31 (2024), p. 161.

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

[GW13] Oded Goldreich and Avi Widgerson. "On derandomizing algorithms that err extremely rarely". In: *Electronic Colloquium on Computational Complexity: ECCC* 20 (2013), p. 152.

[GZ11] Oded Goldreich and David Zuckerman. "Another proof that $BPP \subseteq PH$ (and more)". In: *Studies in complexity and cryptography*. Vol. 6650. Lecture Notes in Comput. Sci. Springer, Heidelberg, 2011, pp. 40–53.

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

[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.

[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.

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

[LOZ22] Zhenjian Lu, Igor C. Oliveira, and Marius Zimand. "Optimal Coding Theorems in Time-Bounded Kolmogorov Complexity". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, Paris, France, July 4-8, 2022*. Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 92:1–92:14. DOI: 10.4230/LIPICS.ICALP.2022.92. URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.92.

[LP22] 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.

[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.

[MV05] Peter Bro Miltersen and N. V. Vinodchandran. "Derandomizing Arthur-Merlin games using hitting sets". In: *Computational Complexity* 14.3 (2005), pp. 256–279.

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

[Raz05] Ran Raz. “Extractors with weak random seeds”. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. Ed. by Harold N. Gabow and Ronald Fagin. ACM, 2005, pp. 11–20. DOI: 10.1145/1060590.1060593. URL: https://doi.org/10.1145/1060590.1060593.

[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.

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

[Sip88] Michael Sipser. “Expanders, randomness, or time versus space”. In: *Journal of Computer and System Sciences* 36.3 (1988), pp. 379–383.

[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.

[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.

[Tel22] Roei Tell. “Quantified derandomization: how to find water in the ocean”. In: *Foundations and Trends® in Theoretical Computer Science* 15.1 (2022), Paper No 1, 125.

[Tel24] Roei Tell. “On defining PPT-search problems and PPT-optimization problems”. In: *Electronic Colloquium on Computational Complexity: ECCC* 31 (2024), p. 163.

[TV12] Terence Tao and Van Vu. *Additive Combinatorics*. Cambridge University Press, 2012.

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

[Wil16] Richard Ryan Williams. “Strong ETH breaks with Merlin and Arthur: short non-interactive proofs of batch evaluation”. In: *Proc. 31st Annual IEEE Conference on Computational Complexity (CCC)*. Vol. 50. 2016, 2:1–2:17.

[XZ25] Zhiyang Xun and David Zuckerman. “Near-optimal averaging samplers and matrix samplers”. In: *Proc. 40th Annual IEEE Conference on Computational Complexity (CCC)*. Vol. 339. 2025, Art. No. 6, 28.

[Zuc06] David Zuckerman. "Linear degree extractors and the inapproximability of max clique and chromatic number". In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 2006, pp. 681–690. URL: https://theoryofcomputing.org/articles/v003a006/v003a006.pdf.

[Zuc97] David Zuckerman. "Randomness-optimal oblivious sampling". In: *Proceedings of the Workshop on Randomized Algorithms and Computation (Berkeley, CA, 1995)*. Vol. 11. 4. 1997, pp. 345–367.

A On Overhead-Free Derandomization

Recall that the non-uniform derandomization of T-time algorithms by Adleman [Adl78] incurs overhead $T \mapsto O(n \cdot T)$, and that this was matched by a uniform algorithm under hardness assumptions (see [CT21b], following [DMO+20]).

As pointed out in prior works [Wil16; CT21b; CT21a; CT23a], there is non-trivial evidence against the possibility of derandomization with time overhead $\ll n \cdot T$. However, we argue that this evidence is far from decisive, and suggest to treat the optimal overhead for derandomization as *a major open problem that requires further study*.

In this short appendix we describe the known evidence against derandomization with time overhead $\ll n \cdot T$. Before doing so, we mention the evidence *for* the possibility that such derandomization is possible, and in fact that *overhead-free* derandomization (i.e., with time overhead $T \mapsto O(T)$) is possible. Indeed, this evidence is at the core of the current work, and we believe should be given at least as much weight as the evidence against this possibility.

> **Evidence for the possibility of overhead-free derandomization:** The question of whether derandomization with overhead $\ll n \cdot T$ is possible reduces to the question of whether targeted samplers with error δ can have sample size $D \ll \log(1/\delta)$ (and linear running time), beating general-purpose samplers.
>
> This possibility is appealing because we are comparing a computational requirement that refers to a single and explicitly given efficient procedure to an *information-theoretic* requirement that refers to all possible functions (including computationally infeasible functions).[53]

[53] To emphasize the point, a typical comparison is of "black-box" objects whose requirement is computational but refers to an entire class (e.g., a PRG that fools an entire circuit-class) with "non-black-box" objects whose requirement is computational and refers to a single, explicitly given procedure (e.g., a targeted PRG that fools only a given procedure). The point is that in our setting the difference is more extreme: we compare an object with an *information-theoretic requirement* (i.e., a general-purpose sampler)

Fast derandomization gives fast non-deterministic counting. As proved by Williams [Wil16, Corollary 3.1] (see [CT21b] for an extension), overhead-free derandomization would yield a non-deterministic algorithm solving #CircuitSAT for formulas of size $2^{o(n)}$ in time $2^{(1/2+o(1))\cdot n}$.[54] The latter task seems hard, but we suggest giving weight to the possibility that such a non-deterministic algorithm might actually exist.

A natural hard candidate. Williams' result puts forward polynomial identity testing as a natural candidate for a problem hard for derandomization (i.e., that forces derandomization overhead). Another hard candidate was posed by Chen and Tell [CT23b], called Time-Bounded Approximate Majority Problem (TAMP).

In TAMP we are given $n/\log(n)$ descriptions of machines (each description of size $\log(n)$), where each machine runs on an empty input in time T and produces a bit. Our goal is to decide whether at most $1/3$ of the machines output zero, or at least $2/3$ of the machines output one. The problem is easy to solve with randomness in time $O(T)$, but it is unclear how to solve it deterministically without executing $\Omega(n)$ of the machines, each for time T and independently of each other.

This hard candidate is natural and appealing. There is currently no evidence that supports its hardness (except for the intuitive appeal).

The non-explicit construction. Recall that Adleman's [Adl78] non-explicit construction yields derandomization with overhead $T \mapsto O(n \cdot T)$. Given that often "random is best", one may conjecture that Adleman's construction is optimal.

Of course, this is unsound. Adleman's construction is a PRG for algorithms with *any* possible n-bit input (considered as non-uniform advice), and it is well-known that derandomization with such PRGs (i.e., "black-box" derandomization) must incur time overhead of approximately n. One may ask what is the best non-explicit non-black-box derandomization, and it is trivial that such derandomization incurs no overhead.[55]

Known constructions and failure of random functions. In [CT21a, Theorem 1.7], the assumption was non-batch-computability over all polynomial-time samplable distributions, and the conclusion was derandomization without overhead that succeeds over all polynomial-time samplable distributions (they called this "free lunch derandomization"). If one assumes that non-batch-computability holds over *all* inputs, the conclusion becomes overhead-free *worst-case* derandomization.

Why then should we not assume non-batch-computability over all inputs? Or, alternatively, hardness of circuits with low-degree gates (as in Theorem 6.3) over all

with a non-black-box object whose requirement is computational and refers only to a single explicitly given procedure (i.e., a targeted sampler).

[54] In prior works, this evidence was presented as refuting #NSETH (i.e., the strong non-deterministic exponential-time hypothesis). Given the recently claimed refutation of #ETH by OpenAI, we fall back to presenting this evidence as giving the #CircuitSAT algorithm.

[55] That is, it is trivial that for every algorithm and input there *exists* a good random string (i.e., a sample of size one) that can be hard-wired, without any overhead.

inputs? One reason is that this assumption fails a standard sanity check: a random function with the required parameters does not seem to be hard over all inputs (but it is hard over uniformly-samplable distributions). This is because the output size (or the circuit size, in case of circuits with low-degree gates) is sub-linear, so the amount of information does not suffice for a union-bound over 2^n inputs. Needless to say, this is not conclusive evidence, and moreover it might be an artifact of specific techniques.

B Another Construction of a Condenser from Sub-constant Rate

In this appendix, we give a construction of a condenser similar to the one stated in Theorem 4.13 and with only slightly different parameters, which is of independent interest. Specifically, in this construction we will use a basic somewhere-condenser that is different than the one in Lemma 4.11, while the framework of repeatedly applying the basic construction remains the same.

B.1 Overview

We start with a somewhere-condenser construction ([Raz05], Theorem 10) from which our main idea originates.

In Raz's construction, he first divides the input source X into several disjoint slices $X_1, \ldots, X_k \in \{0,1\}^n$, then compute some Y s.t. $\mathsf{MExt}(X_1, \ldots, X_k, Y) = 0^n$ where MExt is a multiple-source extractor from [BIW06] and finally outputs $(X_1, \ldots, X_k, Y)$. Additionally, MExt satisfies the following property: for arbitrary $x_1, \ldots, x_k \in \{0,1\}^n$, there exists a unique $y \in \{0,1\}^n$ s.t. $\mathsf{MExt}(x_1, \ldots, x_k, y) = 0^n$.

However, this construction does not work well in our use case: for subconstant-rate inputs, the construction of MExt requires a superconstant number of input sources, which leads to an extra shrinkage of the length of each output source. Computation shows that, as the number of rounds increases, the length of each output source will be reduced to $o(1)$ before achieving a constant rate.

We noticed that, in the proof, a multiple-source extractor is unnecessarily strong for the task. Instead, a multiple-source condenser suffices, as it requires only a constant number of sources for subconstant-rate inputs. Moreover, the proof does not require "condensing" the input sources – it only needs a function $f : (\{0,1\}^n)^k \to \{0,1\}^m$ satisfying the following two properties: 1) the output's min-entropy is slightly larger than the minimum min-entropy over all inputs; 2) for any $x_1, \ldots, x_{k-1} \in \{0,1\}^n$ there exists a unique $y \in \{0,1\}^n$ s.t. $f(x_1, \ldots, x_{k-1}, y) = 0^m$. (We call such f as an algebraic entropy combiner.) This insight enables us to use simpler, weaker, yet more efficient constructions. [56] We construct such a function by weakening the MExt construction

[56] This idea is similar to Remark 5.4 in [BKS+10]. While their construction, $(x_1, \ldots, x_k) \mapsto (x_1, \ldots, x_k, \mathsf{MExt}(x_1, \ldots, x_k))$, differs slightly from Raz's and involves a more complex proof, they observed that an extractor is unnecessarily powerful and introduces additional dependence on the input source rate δ.

from [BIW06].

The roadmap is, we use an algebraic entropy combiner to build a basic somewhere-condenser bSCon, recursively apply bSCon to get the final somewhere-condenser SCon.

B.1.1 Algebraic Entropy Combiner

The object named *entropy combiner* is a function that takes multiple independent sources and outputs a single source with min-entropy slightly larger than the minimum min-entropy of the input sources. We note that this object is a weakened version of a multiple-source condenser.

Lemma B.1 (Algebraic Entropy Combiner)**.** *For every constant $\delta \in (0,1)$, there exists $\epsilon = O(\delta)$, an integer $\ell = 3\lceil(1+\epsilon)/\epsilon\rceil$, a function $p : \mathbb{N} \to \mathbb{N}$ such that $n \le p(n) < 2n$ and a function family $\mathsf{EComb} = \left\{\mathsf{EComb}_n : (\{0,1\}^n)^\ell \to \{0,1\}^{p(n)}\right\}_{n\in\mathbb{N}}$ such that the following holds.*

For every $n \in \mathbb{N}$, and every collection of ℓ independent distributions $X_1, \ldots, X_\ell$ over $\{0,1\}^n$, the output distribution $\mathsf{EComb}_n(X_1, \ldots, X_\ell)$ is γ-close to having min-entropy m' where

$$m' = \min\left\{(1+\epsilon)\cdot\min_{i\in[\ell]}\{H_\infty(X_i)\}, (1-\delta)n\right\}, \quad \gamma = 2^{-(1+\epsilon)m} .$$

Moreover, for any $x_1, \ldots, x_{\ell-1} \in \{0,1\}^n$ there exists a unique $x_\ell \in \{0,1\}^n$ such that $\mathsf{EComb}_n(x_1, \ldots, x_\ell) = 0^{p(n)}$, which we denote by $x_\ell = f_{\mathsf{EComb}}(x_1, \ldots, x_{\ell-1})$. Both EComb and f_{EComb} can be computed in time $\tilde{O}(n)$.

The construction of the algebraic entropy combiner requires the following two lemmas: the first provides a basic multiple-source condenser, and the second reduces the error of the basic construction.

Lemma B.2 (a known generalization of a result from [BIW06])**.** *There exists a real constant $C > 0$ such that for every constant $\delta \in (0,1)$ the following holds. For every triplet of independent distributions $\mathcal{A}, \mathcal{B}, \mathcal{C}$ over a field $\mathbb{F} = \mathbb{F}_{2^p}$ where p is a prime, the distribution $\mathcal{A}\cdot\mathcal{B}+\mathcal{C}$ is $2^{-(C\cdot\delta)\cdot m}$-close to having min-entropy at least $\min\{(1+C\cdot\delta)\cdot m, (1-\delta)\cdot\log|\mathbb{F}|\}$, where $m := \min\{H_\infty(\mathcal{A}), H_\infty(\mathcal{B}), H_\infty(\mathcal{C})\}$.*

Proof sketch. The statement is a generalization of [BIW06, Lemma 3.14]. In their result, they require $m \ge \delta\cdot\log|\mathbb{F}|$ (which we do not require) and they only increase the min-entropy up to $0.9\log|\mathbb{F}|$ (rather than up to $(1-\delta)\log|\mathbb{F}|$ for an arbitrary $\delta > 0$, as in our result).

The proof of [BIW06, Lemma 3.14] relies on an additive combinatorics result of Bourgain, Katz, and Tao [BKT04], stated in [BIW06, Theorem 1.4]. This result asserts that for every subset $A \subseteq \mathbb{F} = \mathbb{F}_{2^p}$ such that $|\mathbb{F}|^\delta < |A| < |\mathbb{F}|^{1-\delta}$ it holds that

$$\max\{|A+A|, |A\cdot A|\} \ge |A|^{1+\epsilon}$$

for some constant $\epsilon = \Theta(\delta)$. To prove Lemma B.2 we use the exact same proof as in [BIW06, Lemma 3.14], except that we replace the additive combinatorics result with

a stronger subsequent result, which appeared in [BGK06; TV12].[57] The latter result removes the restriction $|A| > |\mathbb{F}|^{\delta}$, and has an explicit linear dependency of ϵ on δ (i.e., $\epsilon = C \cdot \delta$ for some constant $C > 1$).

Both improvements follow from using the stronger additive combinatorics result. First, the reason [BIW06] required $m \geq \delta \cdot \log |\mathbb{F}|$ was to meet the condition $|A| > |\mathbb{F}|^{\delta}$, and thus we do not need this requirement. Secondly, the reason [BIW06] increased the min-entropy only up to $0.9 \cdot \log |\mathbb{F}|$ rather than to $(1-\delta) \cdot \log |\mathbb{F}|$ for an arbitrary $\delta > 0$ was to get an expansion of A to $|A|^{1+\epsilon}$ where ϵ is explicit (indeed, a universal constant), avoiding the unspecified dependency on δ in the original result of [BKT04]. Since we use the improved additive combinatorics result, we can increase the min-entropy up to $(1-\delta) \cdot \log |\mathbb{F}|$, while getting an expansion of $|A|^{1+C \cdot \delta}$ (which translates into the min-entropy increase $(1 + C \cdot \delta) \cdot m$ in our statement). ■

Lemma B.3 (Error Reduction by Independent Sources over Finite Fields)**.** *For a finite field $\mathbb{F}$, let $\mathcal{D}_1, \ldots, \mathcal{D}_t$ be t independent random variables over $\mathbb{F}$ that are all ϵ-close to having min-entropy k. Then the distribution*

$$\mathcal{D}_1 + \cdots + \mathcal{D}_t$$

is ϵ^t-close to having min-entropy k, where the $+$ operator means sampling one element for each random variable and summing them up.

Proof. Each $\mathcal{D}_i$ can be viewed as a convex combination of two independent distributions: $\mathcal{D}_i = (1-\epsilon)\mathcal{A}_i + \epsilon\mu_i$, where one distribution $\mathcal{A}_i$ has min-entropy k, and another distribution μ_i is arbitrary. In other words, when sampling from $\mathcal{D}_i$, the element comes from $\mathcal{A}_i$ with probability $1-\epsilon$ and from μ_i with probability ϵ.

Consequently, when sampling from $\mathcal{D}_1 + \cdots + \mathcal{D}_t$, the element will come from $\mu_1 + \cdots + \mu_t$ with probability ϵ^t, and with probability $1-\epsilon^t$ there exists $i \in [t]$ such that the ith summand comes from $\mathcal{A}_i$. In the latter case, we observe that the sum of these random variables has min-entropy k if at least one summand has min-entropy k. We thus conclude that the distribution $\mathcal{D}_1 + \cdots + \mathcal{D}_t$ is ϵ^t-close to having min-entropy k. ■

The algebraic entropy combiner is obtained by reducing the error in Lemma B.2 using Lemma B.3.

Proof of Lemma B.1. Let $C > 0$ be the constant in Lemma B.2, let $\epsilon = C \cdot \delta$, and let $\ell = 3\lceil (1+\epsilon)/\epsilon \rceil$. Let $p = p(n)$ be some fixed prime in $[n, 2n)$ (say, the smallest), and identify each string in $\{0,1\}^n$ with an element in $\mathbb{F}_{2^p}$. [58] Define the function

$$\mathsf{EComb}(x_1, \ldots, x_\ell) := \sum_{i=1}^{\ell/3} (x_{3i-2} \cdot x_{3i-1} + x_{3i}) \ ,$$

[57] This result appears explicitly in [TV12, Corollary 2.56 and Remark 2.57], and is credited to [BKT04; BGK06].

[58] Specifically, we first fix a representation of $\mathbb{F}_{2^p}$ using $\{0,1\}^p$, and then for each $x \in \{0,1\}^n$, we identify it with the element that corresponds to $x0^{p-n} \in \{0,1\}^p$.

where the arithmetic is over $\mathbb{F}_{2^p}$.

For every collection of ℓ independent distributions $X_1, \ldots, X_\ell$ over $\{0,1\}^n$ and $m := \min_i \{H_\infty(X_i)\}$, Lemma B.2 asserts that the distribution of each term $(X_{3i-2} \cdot X_{3i-1} + X_{3i})$ is $2^{-\epsilon m}$-close to having min-entropy at least $\min\{(1+\epsilon)m, (1-\delta)\log|\mathbb{F}_{2^p}|\} = \min\{(1+\epsilon)m, (1-\delta)p\}$, and these distributions are independent. Applying Lemma B.3, the sum of these terms is γ-close to having min-entropy $\min\{(1+\epsilon)m, (1-\delta)p\}$, where $\gamma = (2^{-\epsilon m})^{\ell/3} \leq 2^{-(1+\epsilon)m}$.

For the "moreover" part, note that $\mathsf{EComb}(x_1, ..., x_\ell)$ is a linear function in x_ℓ. In particular, for every fixed $x_1, ..., x_{\ell-1}$ there is a unique x_ℓ such that $\mathsf{EComb}(x_1, ..., x_\ell) = 0^p$, namely

$$x_\ell = f(x_1, \ldots, x_{\ell-1}) := -\sum_{i=1}^{\ell/3-1} (x_{3i-2} \cdot x_{3i-1} + x_{3i}) - x_{\ell-2} \cdot x_{\ell-1} \ .$$

The time complexity of finding a prime $p(n) \in [n, 2n)$ is $O(n \cdot \mathrm{polylog}(n)) = \tilde{O}(n)$ by checking the primality of each integer in $[n, 2n)$. Both EComb and f require at most ℓ arithmetic operations over $\mathbb{F}_{2^p}$, so the time complexity of computing each of these two functions is $\tilde{O}(n) \cdot \ell = \tilde{O}(n)$. ■

B.1.2 The Basic Somewhere-Condenser

We show the construction of the basic somewhere-condenser below.

Theorem B.4. *For any constant $\delta \in (0,1)$, there exists $\epsilon = O(\delta)$, an integer $\ell = 3\lceil(1+\epsilon)/\epsilon\rceil$ and a function family $\mathsf{bSCon} = \{\mathsf{bSCon}_n\}_{n\in\mathbb{N}}$ such that*

$$\mathsf{bSCon}_n : \{0,1\}^n \to (\{0,1\}^m)^\ell$$

is computable in time $\tilde{O}(n)$ and satisfies the following. For every sufficiently large $n \in \mathbb{N}$ and every $\rho = \rho(n) \in [0,1]$, the function bSCon_n is a $[\rho \mapsto \rho']^{1:\ell}_{\sim\gamma}$-somewhere-condenser, where

$$m = \left\lceil \frac{n}{\ell - 1} \right\rceil \ , \quad \rho' = \min\left\{\rho \cdot \left(1 + \epsilon^2/8\right), 1 - 2\delta\right\} \ , \quad \gamma = 2^{-\Theta(\rho m)} \ .$$

Proof. We instantiate Lemma B.1 with the given value of δ, fix parameters $\epsilon, \ell = 3\lceil(1+\epsilon)/\epsilon\rceil$ and the function f_{EComb}. Without loss of generality, we assume $\epsilon < \delta$. If $\epsilon \geq \delta$, then we just set $\epsilon = \delta$, and Lemma B.1 still works well with this value. [59]

We assume that $\ell - 1$ divides n first. Let $m = n/(\ell-1)$. We define $\mathsf{bSCon} : \{0,1\}^n \to (\{0,1\}^m)^\ell$ as follows, viewing an input string $x \in \{0,1\}^n$ as a concatenation of $\ell - 1$ shorter strings $x_1, \ldots, x_{\ell-1} \in \{0,1\}^m$:

$$\mathsf{bSCon}_n(x_1, \ldots, x_{\ell-1}) = (x_1, \ldots, x_{\ell-1}, f_{\mathsf{EComb}}(x_1, \ldots, x_{\ell-1})).$$

[59] A technical point is that a smaller ϵ leads to a larger constant ℓ. However, this does not matter since by analyzing the proof of Lemma B.1, a larger ℓ only leads to a smaller error term γ.

Let us prove that $\mathsf{bSCon} = \mathsf{bSCon}_n$ is a somewhere-condenser. Consider an $(n, \rho n)$-source X as the input to bSCon, and without loss of generality assume that X is a flat source with support $D \subseteq \{0,1\}^n$ of size $2^{\rho n}$. Note that bSCon is an injective mapping, since it extends its input $x_1, ..., x_{\ell-1}$ (i.e., by $f_{\mathsf{EComb}}(x_1, \ldots, x_{\ell-1})$). Hence, the output distribution $Y = \mathsf{bSCon}(X)$ is a flat source on some set $R := \mathsf{bSCon}(D) \subseteq \{0,1\}^{m\ell}$ such that $|R| = |D| = 2^{\rho n}$.

For the case where $\rho > 1 - 2\delta$, the rate ρ' we need to prove for the output is $1 - 2\delta$. We assume for the sake of contradiction that $\mathsf{bSCon}(X)$ is not a $(m, (1-2\delta)m)^{1:\ell}$-source. Then, each output source has at most $2^{(1-2\delta)m}$ possible values in its support, and hence the size of $\mathrm{supp}(X)$ is at most $2^{(1-2\delta)m\cdot(\ell-1)} = 2^{(1-2\delta)n}$, contradicting the assumption that $D = \mathrm{supp}(X)$ has size $2^{\rho n} > 2^{(1-2\delta)n}$. We conclude that bSCon satisfies our requirement when $\rho > 1 - 2\delta$.

For the case where $\rho \leq 1 - 2\delta$, let $\rho' = \rho \cdot (1 + \epsilon^2/6)$.[60] Let $\gamma \in [0,1]$ be the minimal value such that for some $t \in \mathbb{N}$ there are distributions $Y_1, \ldots, Y_t$, each of which is an $(m, \rho' m)^{1:\ell}$-source, and an arbitrary distribution μ, for which $Y = \sum_{i\in[t]} c_i \cdot Y_i + \gamma \cdot \mu$. Then:

Fact B.5. *There exist sets $S_1, \ldots, S_\ell \subseteq \{0,1\}^m$ satisfying $|S_i| = 2^{\rho' m}$ such that $\mathrm{supp}(\mu) \subsetneq S = S_1 \times \ldots \times S_\ell$.*

Proof. Assume otherwise. Then, there exist $S_1, \ldots, S_\ell \subseteq \{0,1\}^m$ such that $|S_i| \geq 2^{\rho' \cdot m}$ for some $i \in [\ell]$, and $\mathrm{supp}(\mu) \supseteq S' := S_1 \times \ldots \times S_\ell$. Thus, we can define $\gamma' := \min_{y\in S'}\{\Pr[\mu = y]\}$ and write Y as

$$Y = \sum_{i\in[t]} c_i \cdot Y_i + \gamma \cdot \gamma' \cdot U_{S'} + \gamma \cdot (1-\gamma')\mu'$$

for some distribution μ'. We notice that the flat source $U_{S'}$ is also an $(m, \rho' m)^{1:\ell}$-source, and $\gamma \cdot (1-\gamma') < \gamma$, which contradicts the assumption that γ is minimal. ∎

Next, we argue that the support of μ is small, based on the fact that it is contained in S and that $\mathsf{EComb}(U_S)$ and $\mathsf{EComb}(\mu)$, as distributions, are statistically far; that is:

Claim B.6. *The support of μ satisfies $|\mathrm{supp}(\mu)| \leq 2^{1+m(\rho'\ell-\rho'')}$, where $\rho'' := \min\{(1+\epsilon)\rho', 1-\delta\}$.*

Proof. On the one hand, by the properties of EComb, the output of $\mathsf{EComb}(U_S)$ is $2^{-(1+\epsilon)\rho' m}$-close to a distribution over $\{0,1\}^{p(m)}$ with min-entropy $\rho'' m$, where $\rho'' := \min\{(1+\epsilon)\rho', 1-\delta\}$; hence, at most a $2 \cdot 2^{-\rho'' m}$ fraction of $y \in S$ satisfy $\mathsf{EComb}(y) = 0^{p(m)}$. On the other hand, every element $y \in \mathrm{supp}(\mu)$ satisfies $\mathsf{EComb}(y) = 0^{p(m)}$; this

[60] ρ' is defined with $(1+\epsilon^2/6)$ and not $(1+\epsilon^2/8)$ to deal with the case where $\ell - 1$ does not divide n (see later). Also the $1-2\delta$ term does not appear in this case, since if $\rho' > 1-2\delta$ then we are also done with a better output rate.

is because $\mathrm{supp}(\mu) \subseteq \mathrm{supp}(Y) \subseteq \mathrm{Range}(\mathsf{bSCon})$ and every x in the range of bSCon satisfies $\mathsf{EComb}(x) = 0^{p(m)}$ by definition. Since $\mathrm{supp}(\mu) \subset S$, it follows that:

$$|\mathrm{supp}(\mu)| \leq 2 \cdot 2^{-\rho'' m} \cdot |S| = 2^{1-\rho'' m} \cdot 2^{\rho' m \ell} = 2^{1+m(\rho' \ell - \rho'')} \quad .$$

■

Finally, by the definition of μ, we know for every $y \in \mathrm{supp}(Y)$ we have that $\gamma \cdot \Pr[\mu = y] < \Pr[Y = y]$, which implies that

$$\begin{aligned}
\gamma &< \min_{y \in \mathrm{supp}(Y)} \left\{ \frac{\Pr[Y = y]}{\Pr[\mu = y]} \right\} \\
&\leq \frac{1/|R|}{1/|\mathrm{supp}(\mu)|} \\
&\leq \frac{1/2^{\rho n}}{1/2^{1+m(\rho' \ell - \rho'')}} \\
&\leq 2 \cdot \exp_2(m(\rho' \ell - \rho'') - \rho m(\ell - 1)) \\
&= 2 \cdot \exp_{2^{\rho m}} \left(\ell \cdot \left(1 + \frac{\epsilon^2}{6} \right) - \frac{\rho''}{\rho} - \ell + 1 \right) \\
&= 2 \cdot \exp_{2^{\rho m}} \left(1 + \frac{1}{2}\epsilon + \frac{1}{2}\epsilon^2 - \min \left\{ (1+\epsilon) \left(1 + \frac{\epsilon^2}{6} \right), \frac{1-\delta}{\rho} \right\} \right) \\
&\leq 2 \cdot \exp_{2^{\rho m}} \left(1 + \frac{1}{2}\epsilon + \frac{1}{2}\epsilon^2 - \min \{1+\epsilon, 1+\epsilon\} \right) \\
&\leq 2 \cdot \exp_{2^{\rho m}} \left(-\frac{1}{2}(\epsilon - \epsilon^2) \right) \\
&= 2^{-\Theta(\rho m)},
\end{aligned}$$

where $\exp_x(y)$ denotes x^y, and the inequality eliminating the min operator is by

$$\frac{1-\delta}{\rho} \geq \frac{1-\delta}{1-2\delta} \geq \frac{1-2\delta+\epsilon}{1-2\delta} \geq 1 + \epsilon.$$

For the case where $\ell - 1$ does not divide n, we pad the input with 0's to make its length a multiple of $\ell - 1$. The number of padding zeros is at most $\ell - 1$, which means that the min-entropy rate of the padded source is at least $\rho n/(n + \ell - 1)$. We apply the bSCon construction on the padded source. For sufficiently large n, following the same argument as above, the output has min-entropy rate

$$(1 + \epsilon^2/6) \cdot \rho \cdot \frac{n}{n + (\ell - 1)} \geq (1 + \epsilon^2/8) \cdot \rho.$$

Regarding the time complexity, we note that the construction of bSCon is simply an invocation of f, so bSCon can also be computed in $\tilde{O}(n)$ time. ■

B.1.3 The Somewhere-Condenser Construction

We give the statement of the somewhere-condenser, which is a little different from Theorem 4.13. In the following statement, the condenser maps n-bit inputs to m-bit sources, where $m = n^{1-\eta}$, and condenses min-entropy rate $n^{-\Omega(1)}$ to min-entropy rate $1 - 2\delta$.

Theorem B.7 (another version of Theorem 4.13)**.** *For any constants $\delta, \eta \in (0,1)$ there exists $\xi = O(\eta \cdot \delta^3)$ such that the following holds. There is a function $\mathsf{SCon} = \{\mathsf{SCon}_n\}_{n\in\mathbb{N}}$ such that*

$$\mathsf{SCon}_n : \{0,1\}^n \to (\{0,1\}^m)^\ell \ ,$$

is computable in time $n^{1+O(\delta\eta)}$ and is an $[n^{-\xi} \mapsto 1-2\delta]^{1:\ell}_{\sim\gamma}$-somewhere-condenser, where

$$m = n^{1-\eta}, \quad \ell \le n^{(1+O(\delta))\eta}, \quad \gamma = 2^{-\Theta(n^{1-\eta})} \ .$$

This result is achieved by iteratively applying the basic somewhere-condenser in Theorem B.4 that increases the min-entropy rate of the input by a multiplicative constant factor and produces a constant number of sources.

Proof of Theorem B.7. We instantiate Theorem B.4 with the given δ, and fix corresponding $\epsilon = O(\delta)$ and $\ell_b = 3\lceil(1+\epsilon)/\epsilon\rceil$ (to avoid notation collisions, we use ℓ_b rather than ℓ). We iteratively compose bSCon for $r := \log_{\ell_b - 1}(n^\eta) = \Theta(\frac{\eta}{\log(1/\epsilon)}) \cdot \log(n)$ times, and define Con as the resulting object.

If the input source X is an $(n, \rho n)$ source where $\rho = n^{-\xi}$, applying Lemma 4.8, the output of Con is a vector of $\ell = \ell_b^r = n^{\eta \log_{\ell_b-1} \ell_b} \le n^{(1+1/\ell_b)\cdot\eta}$ sources, each of which is of length $n/(\ell_b - 1)^r = n^{1-\eta}$ [61], and has min-entropy rate

$$\begin{aligned}
\min\left\{\rho \cdot \left(1 + \frac{\epsilon^2}{8}\right)^r, 1-2\delta\right\} &= \min\left\{\rho \cdot n^{\left(\eta \log_{\ell_b-1}(1+\epsilon^2/8)\right)}, 1-2\delta\right\} \\
&= \min\left\{\rho \cdot n^{\xi}, 1-2\delta\right\} \\
&= \min\{1, 1-2\delta\} \\
&= 1-2\delta,
\end{aligned}$$

where we define the constant as $\xi = \eta \log_{\ell_b-1}(1+\epsilon^2/8)$. By Theorem B.4, the somewhere-condenser in the ith layer has input rate $\rho_i := \min\{\rho \cdot (1+\epsilon^2/8)^{i-1}, 1-2\delta\}$, output length $m_i := n/(\ell_b - 1)^i$ and error term $\gamma_i := 2^{-\Theta(\rho_i m_i)}$. The final output length is thus $m = n/(\ell_b - 1)^r = \Theta(n^{1-\eta})$. The total error term, by Lemma 4.8, is still inverse

[61] If we take into account the ceilings omitted here, the output length will become $n^{1-\Theta(\eta)}$, which does not affect the subsequent calculations and applications. So we omit the ceilings for simplicity.

sub-exponential in n:

$$\begin{aligned}\gamma = \sum_{i=1}^{r} \gamma_i &= \sum_{i=1}^{r} 2^{-\Theta(\rho_i m_i)} \\ &\le \sum_{i=1}^{r} \exp_2\left(-\Theta\left(\rho n \cdot \frac{(1+\epsilon^2/8)^{i-1}}{(\ell_b-1)^i}\right)\right) + \sum_{i=1}^{r} \exp_2\left(-\Theta((1-2\delta)m_i)\right) \\ &\le r \cdot \exp_2\left(-\Theta\left(\rho n^{1-\eta} \cdot \left(1+\frac{\epsilon^2}{8}\right)^{r-1}\right)\right) + r \cdot \exp_2\left(-\Theta((1-2\delta)n^{1-\eta})\right) \\ &\le \exp_2(-\Theta(n^{1-\eta})),\end{aligned}$$

where the first inequality follows from $2^{-\min\{a,b\}} \le 2^{-a} + 2^{-b}$, and the last inequality follows from $\rho = n^{-\xi}$ and the definition of ξ. For the lower bound on ξ, subtituting $\ell_b = 3\lceil(1+\epsilon)/\epsilon\rceil \le 4 + 3/\epsilon$ into the expression, we have

$$\begin{aligned}\xi &= \eta \log_{\ell_b - 1}\left(1+\frac{\epsilon^2}{8}\right) \\ &= \Theta\left(\frac{\eta\epsilon^2}{\log(1/\epsilon)}\right) \\ &= O(\eta\delta^3) \quad .\end{aligned}$$

Time Complexity. Regarding the running time of the $\log_{\ell_b-1} n^\eta$-round process, each application of the basic construction on length-n input takes at most $T_{\mathsf{bSCon}}(n) = \tilde{O}(n)$ time. In the ith layer we apply the bSCon construction on ℓ_b^{i-1} input sources, each of length $m_{i-1} = n/(\ell_b-1)^{i-1}$. The total time cost in the ith layer is

$$T_{\mathsf{bSCon}}\left(\frac{n}{(\ell_b-1)^{i-1}}\right) \cdot \ell_b^{i-1} = \tilde{O}\left(\frac{n}{(\ell_b-1)^{i-1}}\right) \cdot \ell_b^{i-1} = \tilde{O}(n) \cdot \left(\frac{\ell_b}{\ell_b-1}\right)^{i-1}.$$

Adding up the time cost for each of the r layers, the total time complexity is bounded by

$$T_{\mathsf{SCon}}(n) = \tilde{O}(n) \cdot \sum_{i=1}^{r}\left(\frac{\ell_b}{\ell_b-1}\right)^{i-1} = \tilde{O}(n^{1+\eta/(\ell_b-1)\ln(\ell_b-1)}) = O(n^{1+O(\delta\eta)}).$$

■

C Deferred Proofs of Technical Claims

In Appendix C.1 we include postponed proofs from Section 4, and in Appendix C.2 we include a postponed proof from Section 7.

C.1 Proofs of claims from Section 4

In this appendix we include proofs that were postponed from Section 4, starting from deferred proofs from Section 4.1. We first present the construction of a somewhere-condenser for subconstant rate, which is obtained by iteratively applying the somewhere-condenser of Zuckerman [Zuc06] from Lemma 4.11.

Theorem C.1 (Somewhere-Condenser for Subconstant-Rate Inputs; Theorem 4.13, restated)**.** *For any constants $\delta, \eta \in (0,1)$, there exists a constant $\xi = O(\delta\eta)$ and a $[\rho \mapsto 1-\delta]^{1:\ell}_{\sim\gamma}$-somewhere-condenser* SCon *for every $\rho = \rho(n) \geq n^{-\xi}$, where*

$$m = n^{1-\eta}, \quad \ell \leq n^{2\eta}, \quad \gamma = 2^{-\Theta(n^{1-\eta})}.$$

Moreover, SCon *can be computed in $O(n^{1+O(\eta)})$ time.*

Proof. Fix δ, η as given parameters. Let $\delta_0 \geq n^{-\xi}$ be the source rate for some ξ we choose later. We compose Lemma 4.12 with itself s times using parameters $c = \delta/2$, $\alpha = O(\delta)$ and $\beta = \alpha^2/32$.

Let n_i, δ_i be the source length and source rate after the ith composition respectively. We have

$$n_i \in [(2/3)^i \cdot n, (2/3)^i(1+\beta)^i \cdot n], \quad \delta_i = \left(\frac{1+\alpha/2}{1+\beta}\right)^i \cdot \delta_0 = (1+O(\alpha))^i \cdot \delta_0.$$

Let $s \in [\eta \log_{1.5} n, \eta \log_{1.5/(1+\beta)} n]$ be such that $n_s \in [m, 1.5m]$ where $m = n^{1-\eta}$. Since $n_i > n_{i+1}$ is strictly decreasing, there exists a maximum integer i such that $n_i > 1.5m$. For the next term n_{i+1}, we know $1.5m > n_{i+1} \geq n_i \cdot (2/3) > m$ falls in the range. Therefore such s exists.

To output a vector of fix-length sources with length m, we truncate each coordinate in the output somewhere-random source from length n_s to m.

The number of sources is bounded by

$$\ell = 2^s \leq n^{\eta/\log(1.5/(1+\beta))} \leq n^{2\eta}.$$

The source rate δ_s is

$$\delta_s = \delta_0 \cdot (1+O(\alpha))^s \geq \delta_0 \cdot n^{\eta \log_{1.5}(1+O(\alpha))} = \delta_0 \cdot n^{O(\delta\eta)} = \Omega(1) \geq 1 - \delta/2$$

by selecting an appropriate $\xi = O(\delta\eta)$ in the restriction $\delta_0 \geq n^{-\xi}$. And the final rate of the output is at least $1 - (1-\delta_s)n_s/m \geq 1 - 1.5 \cdot \delta/2 > 1 - \delta$.

The total error γ is bounded by

$$\gamma \leq \sum_{i=0}^{s-1} \exp_2(-n_i \cdot \alpha\delta_i/20) = 2^{-\Theta(n^{1-\eta})}.$$

We claim that the described construction is the desired $[n^{-\xi} \mapsto 1-\delta]^{1:\ell}_{\sim\gamma}$-somewhere condenser

$$\mathsf{SRC} : \{0,1\}^n \to (\{0,1\}^m)^\ell,$$

where

$$m \in [n^{1-\eta}, n^{1-\eta+O(\delta^2)\eta}], \quad \ell = 2^s = n^{\eta/\log 1.5} \le n^{2\eta}, \quad \gamma \le 2^{-\Theta(n^{1-\eta})}.$$

The running time of SRC in the ith iteration includes the time of running Lemma 4.12 on 2^{i-1} sources, each of length $(2/3)^{i-1}n$. Summing up the cost over s iterations, the total running time is bounded by

$$\sum_{i=1}^{s} 2^{i-1} \cdot \tilde{O}(n_{i-1}) \le n^{1+O(\eta)}.$$

■

Next, we present the proof that combining the merger of Theorem 4.10 with the foregoing somewhere condenser yields a fast condenser for subconstant rate.

Theorem C.2 (Fast Condenser for Subconstant Rate Inputs; Theorem 4.14, restated)**.** *For any two constants $\epsilon, \gamma \in (0,1)$, there exists a constant $\xi = O(\epsilon^3)$ and a function family $\mathsf{Con} = \{\mathsf{Con}_n\}_{n\in\mathbb{N}}$ where*

$$\mathsf{Con}_n\colon \{0,1\}^n \times \{0,1\}^{d=\epsilon\cdot\log(n)+O(1)} \to \{0,1\}^{m=n^{1-\epsilon^2/4}}$$

is an $[n^{-\xi} \mapsto 1-\epsilon]_{\sim\gamma}$-condenser and the multiset of strings $\{\mathsf{Con}_n(x;s) : s \in \{0,1\}^d\}$ is computable in time $O(n^{1+O(\epsilon)})$.

Proof. We instantiate Theorem 4.13 with parameter values $\delta = \epsilon/2$ and $\eta = \epsilon^2/4$. Let $\xi = O(\delta\eta)$ and $m = n^{1-\eta}$ and $\ell \le n^{2\eta}$ and $\gamma_{\mathsf{SCon}} = o(1)$ be the corresponding parameters. We then instantiate Theorem 4.10 with parameter values ($\lambda = \epsilon/2, \gamma_{\mathsf{DW}} = \gamma/2$), consider the merger M_{DW} for somewhere random sources with length m and ℓ coordinates, and the corresponding seed length is

$$d = \lambda^{-1}\log\frac{2\ell}{\gamma_{\mathsf{DW}}} = \lambda^{-1}\log\ell + O(1) = \frac{2\eta}{\epsilon/2}\log n + O(1) \le \epsilon\log n + O(1).$$

Now we define

$$\mathsf{Con}(x;s) = M_{\mathsf{DW}}(\mathsf{SCon}(x);s).$$

Consider an $(n,\rho n)$-source X where $\rho = n^{-\xi}$. By Theorem 4.13, the output distribution $\mathsf{SCon}(X)$ is a convex combination of several independent $(m,(1-\delta)m)^{1:\ell}$-sources $\mathcal{S}_i$, each with weight w_i, and an arbitrary error distribution μ with weight γ_{SCon}. Since M_{DW} is a $[d,(m,b)^{1:\ell} \mapsto (m,(1-\lambda)b) \sim \gamma_{\mathsf{DW}}]$-merger, for each $\mathcal{S}_i$ the distribution $M_{\mathsf{DW}}(\mathcal{S}_i, U_d)$ is γ_{DW}-close to some distribution $\mathcal{D}_i$ that has min-entropy at least

$$(1-\delta)(1-\lambda)\cdot m \ge (1-\epsilon)\cdot m \ .$$

Therefore, $\mathsf{Con}(X;U_d) = M_{\mathsf{DW}}(\mathsf{SCon}(X);U_d)$ is a convex combination of 1) several independent distributions $\mathcal{D}_i$ satisfying $H_\infty(\mathcal{D}_i) \ge (1-\epsilon)\cdot m$, each with weight $(1-\gamma_{\mathsf{DW}})\cdot w_i$, 2) several error distributions μ_i, each with weight $\gamma_{\mathsf{DW}}\cdot w_i$, and 3) an error distribution $M_{\mathsf{DW}}(\mu;U_d)$, with weight γ_{SCon}. The total weight of error distributions is at most

$$\sum_i \gamma_{\mathsf{DW}}\cdot w_i + \gamma_{\mathsf{SCon}} < \gamma_{\mathsf{DW}} + \gamma_{\mathsf{SCon}} < \gamma.$$

Time Complexity. For computing the multiset of strings enumerating over all seeds, the time complexity becomes

$$T_{\mathsf{batch\text{-}Con}}(n) = T_{\mathsf{SCon}}(n) + T_{\mathsf{batch\text{-}DW}}(n^{1-\eta}, n^{2\eta}) = O\left(n^{1+O(\epsilon)}\right).$$

■

Finally, we briefly describe the construction of the DW merger and analyze its time complexity. We do not provide a proof of its correctness, as we have not modified the original construction.

Theorem C.3 ([DW08; DKS+13]). *For any real constant $\lambda > 0$ and $\gamma > 0$[62], there exists an explicit $[\rho \mapsto (1-\lambda)\rho]^{\ell:1}_{\sim\gamma}$-merger $M_{\mathsf{DW}} : (\{0,1\}^n)^\ell \times \{0,1\}^d \to \{0,1\}^n$ for every $n > 0$ and every $\rho = \rho(n) \in [0,1]$, with seed length*

$$d = \lambda^{-1}\log(2\ell/\gamma).$$

Moreover, the multiset of strings $\left\{M_{\mathsf{DW}}(x_1,\ldots,x_\ell;s) : s \in \{0,1\}^d\right\}$ can be computed in time $T_{\mathsf{batch\text{-}DW}}(n,\ell) = \tilde{O}(\ell^3 + n\ell^{1+1/\lambda})$.

Construction. Define $d = \lceil \lambda^{-1}\log(2\ell/\gamma)\rceil$, and let $\mathbb{F}$ be a finite field of size $q = 2^d$. Then

$$(2\ell/\gamma)^{1/\lambda} < q \le 2(2\ell/\gamma)^{1/\lambda}.$$

We assume w.l.o.g. that $n = r \cdot d$ for some r, and treat the input domain as $(\mathbb{F}^r)^\ell \times \mathbb{F}$. Let $e_1,\ldots,e_\ell \in \mathbb{F}$ be ℓ distinct field elements. We define ℓ univariate polynomials $c_1(s),\ldots,c_\ell(s) \in \mathbb{F}[s]$ as follows:

$$c_i(s) := \prod_{j\in[\ell]\setminus\{i\}} \frac{s-e_j}{e_i-e_j}.$$

For $j \in [\ell]$, $c_i(e_j)$ evaluates to 1 iff $i = j$, and for $j \neq i$ we have $c_i(e_j) = 0$. We define the merger function as

$$M_{\mathsf{DW}}(x_1,\ldots,x_\ell;s) := \sum_{i\in[\ell]} c_i(s)x_i.$$

Time Complexity. First, it takes $2^{O(d)}$ time to find the field $\mathbb{F}_q$ (i.e., find an irreducible polynomial of degree d over $\mathbb{F}_2$). Applying of the merger in error reduction needs to compute $M(x_1,\ldots,x_\ell;s)$ for all $s \in \mathbb{F}$, where $x_1,\ldots,x_\ell$ are fixed. We can treat each coordinate of the output of M as a degree-ℓ univariate polynomial $f_i(s) := M(x_1,\ldots,x_\ell;s)_i = \sum_j c_j(s)(x_j)_i$. Computing the polynomial representation of each $c_i(s)$ can be done using $O(\ell^2)$ arithmetic operations over $\mathbb{F}$, and then computing the representation of each $f_i(s)$ takes $O(\ell^2)$ arithmetic operations over $\mathbb{F}$. So

[62]The error term γ can also be subconstant, but here for simplicity we keep it a constant.

within $O(\ell^3 + \ell^2 r)$ arithmetic operations we get $f_i(s)$ for every $i \in [r]$, and each time evaluating $f_i(s)$ takes $O(\ell r)$ arithmetic operations. Hence the total time complexity is bounded by

$$\begin{aligned} T_{\mathsf{batch\text{-}DW}}(n,\ell) &= O(\ell^3 + \ell^2 r)\tilde{O}(d) + |\mathbb{F}| \cdot O(\ell r)\tilde{O}(d) \\ &= \tilde{O}(\ell^3) + \tilde{O}(\ell^2 n) + \tilde{O}\left(n\ell^{1+1/\lambda}\right) \\ &= \tilde{O}\left(\ell^3 + n\ell^{1+1/\lambda}\right). \end{aligned}$$

Next, we state and prove the claim that was stated after Definition 4.17, which asserts that the restriction of circuits to have fan-out-2 is, essentially, without loss of generality (i.e., the only cost is assuming that the function $OUT_i(g,b)$ is efficiently computable).

Proposition C.4 (fan-out-two can be assumed in Definition 4.17 without significant loss)**.** *Let C be a circuit family as in Definition 4.17 but without assuming that NAND gates have fan-out 2. Further assume that the function $OUT(i,g,j) = OUT_i(g,j)$ is computable in polynomial time $\mathrm{polylog}(T)$.*[63] *Then, there is another circuit family C' of size $T' = d \cdot \tilde{O}(T)$ and depth $d \cdot \mathrm{polylog}(T)$ that satisfies the requirements of Definition 4.17 (including the fan-out-2 restriction).*

We provide a proof sketch and omit low-level technical details that are cumbersome yet straightforward.

Proof sketch. We first describe the construction of C'_n, and then argue that it has the required uniformity conditions (which will follow easily by inspection).

The structure of C'_n imitates the structure of C_n, except that between each two layers we add $\mathrm{polylog}(T)$ auxiliary layers, each of size at most $\tilde{O}(T)$. Specifically, after each layer i that originally existed in C_n, the sequence of auxiliary layers in C'_n performs the operations detailed below. Intuitively, the operations are intended to copy the value of each gate g for $\Delta(g) = \text{fan-out}(g)$ times, and then sort the values by their output destinations to feed them into the next layer in an easily-computable way.

Compute fan-outs. For every gate g, compute $\Delta(g)$ to obtain $O(\log T)$ gates in an auxiliary layer whose values are $(g, v_g, \Delta(g))$ (where v_g is the value of g in the circuit). Specifically, the auxiliary layers computing $\Delta(g)$ perform a binary search for $\log(T)$ rounds using the function $OUT_i(g,j)$ (i.e., trying out values of j in the interval $[0,T]$).

Compute the starting positions of each gate. For every gate g (now represented as $(g, v_g, \Delta(g))$) compute $p(g) = \sum_{g_0 < g} \Delta(g_0)$ (i.e., the starting position of the values of g in the sequence of values of all gates, each of which copied suitably many times), to obtain $O(\log T)$ gates whose values are $(g, v_g, p(g))$.

[63] Specifically, $OUT(i,g,j)$ takes a layer index i and an index g of a gate in that layer and $j \in [T]$, and returns either the jth neighbor of g if g has at least j neighbors, or $\perp$.

This is done in depth polylog(T) by a tree that recursively pairwise-sums the values $\{\Delta(g)\}_g$. Importantly, instead of having one tree per gate (which would cause a quadratic size blow-up), we use a single summing-tree for the entrie circuit, and "branch out" the values $p(g)$ of each g as needed from this single tree.

Preparing the array. The goal in this step is to create an array in which the value of each g appears in location $p(g)$. Specifically, the circuit first creates a gate-array with $2T$ blocks of size $O(\log T)$, that includes one block per gate g (containing $(g, v_g, p(g))$) and additional T "slot" blocks with $O(\log T)$ empty space (the slot blocks will be used later for storing copied values of each gate).

To do so, the circuit uses a sorting network of size $\tilde{O}(T)$ whose internal connectivity is easy to compute (e.g., bitonic sort a-la [BCT25] and Theorem 4.23) to sort the $2T$ blocks, where each slot-block has a key in $[T]$ and each gate-block has key $p(g)$. Note that after this sorting step, the block of each g appears after precisely $p(g)$ slot-blocks, with $\Delta(g)$ empty slot-blocks after it.

Copying gate-values across the array. The goal in this step is to copy the value of each g to the $\Delta(g)$ empty slot-blocks after it. Let us explain how to do this in parallel.

The circuit does this in $\log(T)$ iterations. In each iteration, each slot-block will store a gate-value and an index that represents the position $p(g)$ of the gate whose value it currently stores. The circuit initializes these as $(v_g, p(g))$ for slot-blocks immediately after a gate-block of a gate g; and as $(\perp, -1)$ for other slot-blocks. Then, for $i = 1, 2, 4, ..., 2^{\log T}$, each slot-block B_k in position k examines the slot-block B_{k-i} in position $k - i$, and either B_k retains its existing stored values (if the position written at B_{k-i} is smaller than the position written in B_k), or copies both the gate-value and the position stored in slot-block B_{k-i} (i.e., if the position at $k - i$ is larger than the position at k). Note that after i iterations, B_k stores the gate-value and position corresponding to the largest position that was initially stored in blocks in the interval $[k - 2^i, k]$. Hence (and because we initially placed each gate g at position $p(g)$ and copied $p(g)$ to the subsequent slot-block), after $\log(T)$ iterations, each gate-value is copied $\Delta(g)$ times.

Putting gate-values beneath the gates they feed. This step imitates the construction in [BCT25]. Specifically, the circuit's goal now is to create a gate-array of $[2T]$ binary values, and for each gate $\bar{g}$ in the *next* non-auxiliary layer that is fed by gates g, g' in the *current* non-auxiliary layer (i.e., the layer above which we are adding auxiliary layers), place values v_g and $v_{g'}$ in locations $(\bar{g}, 0)$ and $(\bar{g}, 1)$.

This is done by computing $OUT(i, g, j)$ for each copy j of each gate g (and adding the corresponding information to the block), and then using a sorting network to place the value below $OUT(i, g, j)$. At this point, each $\bar{g}$ takes values from the gates directly below it. The details are essentially identical to [BCT25, Sections 2.1.2 & 4.1].

Complexity. The main thing we wanted to achieve is fan-out 2, and this follows by inspection (and using a sorting network that has this feature, where again bitonic

sort suffices). Also note that the depth of C'_n increased by a multiplicative factor of $\mathrm{polylog}(T)$ (i.e., the number of auxiliary layers between each pair of layers), and since each auxiliary layer has size $\tilde{O}(T)$ (using an efficient sorting network), the size blow-up is a multiplicative factor of $\tilde{O}(T)$.

As for preserving uniformity, the circuit as a whole can still be printed in time $\tilde{O}(T)$, since each auxiliary layer can be printed in such time. And the connectivity can be decided in polylogarithmic time by inspection of each part, and using an efficient sorting network (indeed, again, bitonic sort suffices). ■

C.2 Proof of the coding theorem from Lemma 7.3

We provide a detailed proof of Lemma 7.3 in this section. The proof relies on variants of [LOZ22, Theorem 5] and [GKL+22, Lemma 21] that refer to cpKt and cpK^t complexity, as stated below. The proofs of these lemmas are refined from and almost identical to the original proofs.

We first introduce the notion of cpK^t which helps to prove the lemma.

Definition C.5 (cpK^t)**.** *For every $\delta \in (0,1)$, $t : \mathbb{N} \to \mathbb{N}$, $w \in \{0,1\}^*$ and $x \in \{0,1\}^*$, the conditional time-bounded probabilistic Kolmogorov complexity of w given advice string x, time bound t and confidence δ is defined as*

$$\mathsf{cpK}^t_\delta(w|x) := \min_\ell \left\{ \ell \in \mathbb{N} : \Pr_{r\in\{0,1\}^{t(|w|)}} \left[\exists \Pi \in \{0,1\}^{\le \ell} : U(\Pi(x;r), 1^{t(|w|)}) = w \right] \ge \delta \right\}.$$

Lemma C.6 (A variant of [LOZ22, Theorem 5] on cpK^t)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^n; x)$ runs in time $T(n) \ge n$ and outputs a string $w \in \{0,1\}^n$ with probability at least $\delta > 0$ for some string $x \in \{0,1\}^*$. Then*

$$\mathsf{cpK}^{\mathrm{poly}(T(n))}_{2/3}(w|x) \le \log(1/\delta) + O(\log T(n)) + O_{|A|}(1),$$

and the constants behind $O(\log T(n))$ and $\mathrm{poly}(T)$ are independent of the description size $|A|$ of A.

Lemma C.7 ([LOZ22], Lemma 31)**.** *For any $T \in \mathbb{N}$ and $\delta \in (0,1]$, there exists a family of functions*

$$\mathcal{H} = \left\{ H_r : \{0,1\}^\ell \to \{0,1\}^T \right\}_{r\in\{0,1\}^k}$$

where $k = \mathrm{poly}(T)$ and $\ell = \log(1/\delta) + O(1)$ such that the following holds. Let $M : \{0,1\}^T \to \{0,1\}^*$ be a function computable in time T and let $x \in \mathrm{Range}(M)$ be such that

$$\Pr_{z\sim\{0,1\}^T}[M(z) = x] \ge \delta.$$

It holds that

$$\Pr_{r\sim\{0,1\}^k}[\exists z \in \{0,1\}^\ell : M(H_r(z)) = x] \ge 2/3.$$

Moreover, given $r \in \{0,1\}^k$ and $z \in \{0,1\}^\ell$, $H_r(z)$ can be computed in time $\mathrm{poly}(T)$.

Proof of Lemma C.6. Let us view $M := A(1^n; x)$ as a function that takes $T := T(n)$ random bits and outputs $w \in \{0,1\}^n$ with probability at least δ. By Lemma C.7, for at least $2/3$ of $r \in \{0,1\}^{\mathrm{poly}(T)}$, we get a function $H_r : \{0,1\}^\ell \to \{0,1\}^T$, where $\ell = \log(1/\delta) + O(1)$, such that $M(H_r(z)) = w$ for some $z \in \{0,1\}^\ell$. Moreover, given r and z, $H_r(z)$ can be computed in $\mathrm{poly}(T)$ time. This means that for at least $2/3$ of $r \in \{0,1\}^{\mathrm{poly}(T)}$, there is some advice string $\alpha \in \{0,1\}^{\log(1/\delta)+O(\log T)+O_{|A|}(1)}$, which encodes T, the code of A, the code for computing H_r using r, and some good $z \in \{0,1\}^\ell$, such that using α together with r and x we can recover w in time $\mathrm{poly}(T)$. This implies

$$\mathsf{cpK}^{\mathrm{poly}(T)}_{2/3}(w|x) \le \log(1/\delta) + O(\log T) + O_{|A|}(1).$$

■

The proof of the following lemma is identical to the original lemma, so we omit it.

Lemma C.8 (A variant of [GKL+22, Lemma 21] on cpK^t)**.** *For any string $w, x \in \{0,1\}^*$, time bound $t \in \mathbb{N}$, and $0 \le \alpha < \beta \le 1$, we have*

$$\mathsf{cpK}^{O(qt/\alpha)}_\beta(w|x) \le \mathsf{cpK}^t_\alpha(w|x) + O(\log(q/\alpha)),$$

where $q := \ln(1/(1-\beta))$.

Now we prove the coding lemma for cpKt complexity.

Lemma C.9 (Lemma 7.3, restated)**.** *Suppose there is a randomized algorithm A for sampling strings such that $A(1^n; x)$ runs in time $T(n) \ge n$ and outputs a string $w \in \{0,1\}^n$ with probability at least $\delta > 0$. Then for every $0 < \beta \le 1$,*

$$\mathsf{cpKt}_\beta(w|x) \le \log(1/\delta) + O(\log T(n)) + O(\log\log 1/(1-\beta)) + O_{|A|}(1),$$

and the constants behind $O(\log T(n))$ and $\mathrm{poly}(T)$ are independent of the description size $|A|$ of A.

Proof. Applying Lemma C.6, we know

$$\mathsf{cpK}^{\mathrm{poly}(T)}_{2/3}(w|x) \le \log(1/\delta) + O(\log T(n)) + O_{|A|}(1)$$

Let $q := \ln(1/(1-\beta))$. Amplifying the success probability using Lemma C.8,

$$\begin{aligned}\mathsf{cpK}^{O(q\cdot\mathrm{poly}(T))}_\beta(w|x) &\le \mathsf{cpK}^{\mathrm{poly}(T)}_{2/3}(w|x) + O(\log 3q/2)\\ &\le \log(1/\delta) + O(\log T) + O(\log q) + O_{|A|}(1).\end{aligned}$$

Finally, by the definition of cpKt and cpK^t, we know

$$\mathsf{cpKt}_\beta(w|x) \le \mathsf{cpK}^t_\beta(w|x) + \log t$$

for every time bound t. Therefore,

$$\begin{aligned}\mathsf{cpKt}_\beta(w|x) &\leq \mathsf{cpK}_\beta^{O(q\cdot\mathrm{poly}(T))}(w|x) + \log O(q\cdot\mathrm{poly}(T)) \\ &\leq \log(1/\delta) + O(\log T) + O(\log q) + O_{|A|}(1).\end{aligned}$$

■

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il