Electronic Colloquium on Computational Complexity, Report No. 97 (2022)

Unstructured Hardness to Average-Case Randomness

Lijie Chen [*] Ron D. Rothblum [†] Roei Tell [‡]

July 3, 2022

Abstract

The leading technical approach in uniform hardness-to-randomness in the last two decades faced several well-known barriers that caused results to rely on overly strong hardness assumptions, and yet still yield suboptimal conclusions.

In this work we show uniform hardness-to-randomness results that *simultaneously break through all of the known barriers*. Specifically, consider any one of the following three assumptions:

1. For some $\epsilon > 0$ there exists a function f computable by uniform circuits of size $2^{O(n)}$ and depth $2^{o(n)}$ such that f is hard for probabilistic time $2^{\epsilon \cdot n}$.
2. For every $c \in \mathbb{N}$ there exists a function f computable by logspace-uniform circuits of polynomial size and depth n^2 such that every probabilistic algorithm running in time n^c fails to compute f on a $(1/n)$-fraction of the inputs.
3. For every $c \in \mathbb{N}$ there exists a logspace-uniform family of arithmetic formulas of degree n^2 over a field of size $\mathrm{poly}(n)$ such that no algorithm running in probabilistic time n^c can evaluate the family on a worst-case input.

Assuming any of these hypotheses, where the hardness is for every sufficiently large input length $n \in \mathbb{N}$, we deduce that $\mathcal{RP}$ can be derandomized in *polynomial time* and on *all input lengths*, on average. Furthermore, under the first assumption we also show that $\mathcal{BPP}$ can be derandomized in polynomial time, on average and on all input lengths, with logarithmically many advice bits.

On the way to these results we also resolve two related open problems. First, we obtain an *optimal worst-case to average-case reduction* for computing problems in linear space by uniform probabilistic algorithms; this result builds on a new instance checker based on the doubly efficient proof system of Goldwasser, Kalai, and Rothblum (J. ACM, 2015). Secondly, we resolve the main open problem in the work of Carmosino, Impagliazzo and Sabin (ICALP 2018), by deducing derandomization from weak and general fine-grained hardness hypotheses.

[*]Massachusetts Institute of Technology, MA. Email: wjmzbmr@gmail.com

[†]Technion, Israel. Email: rothblum@cs.technion.ac.il

[‡]The Institute of Advanced Study at Princeton NJ and the DIMACS Center at Rutgers University, NJ. Email: roei.tell@gmail.com

ISSN 1433-8092

Contents

1 Introduction

A classical line of work in complexity theory is focused on *uniform hardness vs randomness* results. These are results that connect lower bounds for uniform probabilistic algorithms to average-case derandomization. For example, as proved in the classical result of Impagliazzo and Wigderson [IW98], if $\mathcal{BPP} \neq \mathcal{EXP}$, then $\mathcal{BPP}$ can be simulated in sub-exponential time, on average and infinitely often.[1]

In contrast to works concerning *non-uniform* hardness vs randomness (cf., e.g., [NW94; IW97; STV01; SU05; Uma03]), the currently known results for uniform hardness vs randomness seem suboptimal. For comparison, recall that in the non-uniform setting we know that $\mathcal{E} = \mathcal{DTIME}[2^{O(n)}]$ is hard for circuits of size $s(n)$ if and only if there exists a pseudorandom generator (PRG) for linear-sized circuits with seed length linear in $s^{-1}(\text{poly}(n))$ (see [Uma03]); in particular, for the "high-end" regime, $\mathcal{E}$ is hard for circuits of size $2^{\epsilon \cdot n}$ if and only if there exists such a PRG with seed length $O(\log n)$.[2] However, for *uniform* hardness vs randomness, the results that we know do *not* scale to the "high-end" regime. In fact, even if we assume hardness for *specific* functions in $\mathcal{SPACE}[O(n)]$ that are conductive for these results, rather than hardness for arbitrary functions in $\mathcal{E} = \mathcal{DTIME}[2^{O(n)}]$, we still only know how to deduce average-case derandomization in super-polynomial time $n^{\text{polyloglog}(n)}$ (see [CRT+20]).

As one might expect, this classical challenge attracted considerable interest over the years. The main focus was improving the parameters of the hardness vs randomness tradeoff, trying to deduce faster average-case derandomization from as weak a hardness hypothesis as possible (see, e.g., [CNS99; Kab01; TV07; GV08; CRT+20]). Parallel lines of work studied extensions of this paradigm to derandomization of proof systems, in which case we can obtain worst-case derandomization under uniform hardness assumptions for the corresponding class of protocols (see, e.g., [Lu01; GSTS03; SU07]); and to derandomization that relies on fine-grained hardness hypotheses for specific functions in $\mathcal{P}$, in which case we can circumvent some of the barriers above, and obtain average-case derandomization in polynomial time and on all input lengths (see [CIS18]). The known results have been widely applied throughout complexity theory (for some applications see, e.g., [ABK+06; San09; KKO13; OS17; IKV18]).

The main technical challenges and obstacles. Let us explain the main challenge that has been obstructing progress so far. All the results described above (with the exception of [CIS18]) rely on reconstructive PRGs. Loosely speaking, these are generators that transform a "hard" truth-table f into a set of pseudorandom strings, and the proof of correctness relies on a reduction: A distinguisher for a pseudorandom set is converted into an efficient procedure for the hard function. Indeed, there are two parts in such a construction, the generator and the reconstruction procedure.

When starting from a strong hardness assumption, such as hardness for non-uniform circuits, the reconstruction procedure may use "strong" resources, such as non-uniformity. In contrast, when only assuming hardness for uniform probabilistic algorithms, the reconstruction procedure must also be a uniform algorithm. Alas, we currently *do not know* how to construct efficient generators with uniform and efficient reconstruction procedures when the "hard" truth-table f is an arbitrary function in

[1] The precise meaning of "on average" in this result is that for every $L \in \mathcal{BPP}$ and $\epsilon > 0$ there exists $L' \in \mathcal{DTIME}[2^{n^\epsilon}]$ such that for every polynomial-time samplable collection of distributions $\mathbf{x} = \{\mathbf{x}_n\}_{n \in \mathbb{N}}$ and for infinitely many n's it holds that $\Pr_{x \sim \mathbf{x}_n}[L'(x) = L(x)] \geq 1 - 1/n$.

[2] Throughout the exposition, we always assume that the running time of a PRG is exponential in its seed length. (This is because in the derandomization application we enumerate over the seeds of a PRG.)

$\mathcal{E}$. This is because known ideas for reconstruction procedures rely on *specific structural properties* of f, namely that it is downward self-reducible and randomly self-reducible;[3] such structural properties exist only for functions in $\mathcal{PSPACE}$. (For details see the classical work [IW98], and for further "barrier" results see [GV08].)[4]

The situation gets even worse: Since we need f to admit the specific structural properties mentioned above, we cannot obtain derandomization from hardness of an *arbitrary* function in the relevant class. This leaves us with two choices – either assume hardness for specific functions, which seems an overly narrow assumption; or reduce arbitrary functions in the class to complete functions that admit the structural properties, which typically yields super-polynomial derandomization overheads. Moreover, the known $\mathcal{PSPACE}$-complete functions that admit the required structural properties can be computed in time $2^{o(n)}$, and thus are not sufficiently hard to yield derandomization in polynomial time. And to top this off, known results yield derandomization that succeeds only on infinitely many input lengths (two exceptions are [CIS18; CRT+20]).

Our contributions, in a gist. In this work we show how to *simultaneously bypass all of the obstacles mentioned above*. Specifically, we will show uniform hardness vs randomness results that:

1. Rely on hardness for functions that **do not admit the structural properties** that were required for previous results. In particular, our results start from hardness for functions that are not necessarily computable in $\mathcal{PSPACE}$.

2. Do not need to assume hardness for a specific function: It suffices to assume **hardness for any function in the relevant class** (without causing overheads in the running time of the derandomization algorithm).

3. Can yield derandomization that works in **polynomial time**, assuming that a function in the relevant class is sufficiently hard.

4. Yield derandomization algorithms that work **on all input lengths**, rather than only on infinitely many inputs lengths.

The downside of our results is that we will either derandomize $\mathcal{RP}$ (i.e., probabilistic algorithms with one-sided error), or derandomize $\mathcal{BPP}$ using a small number of advice bits (e.g., logarithmically many or less).

The main idea allowing us to break through the former obstacles is to *rely on machinery constructed for non-black-box derandomization* in the very recent work of two of the authors [CT21]. This machinery was previously used in a different context: In the previous work the hypothesized hardness was very strong, namely hardness on *almost all inputs*,[5] and the conclusion was a worst-case derandomization of $\mathcal{BPP}$. In contrast, in the current work we adapt this machinery to work with the weaker *standard notions of worst-case hardness*, and conclude *average-case derandomization* of $\mathcal{RP}$ and $\mathcal{BPP}$.

[3] Recall that a function f is downward self-reducible if we can compute f quickly (say, in small polynomial time) at any given n-bit input when given oracle access to f at inputs of length $n-1$. A function f is randomly self-reducible if we can quickly evaluate f at any given n-bit input, with high probability, given access to evaluations of f at random n-bit inputs.

[4] These obstacles were bypassed in the original work of [IW98] by a specific argument that introduced *significant time overheads*. Specifically, to obtain a PRG with seed length $s^{-1}(n)$ their result needs hardness for probabilistic algorithms running in time approximately $s(s(n))$; see [TV07, Section 1.2] for details.

[5] That is, the hard function had multiple output bits and every probabilistic algorithm running in time (say) n^{100} failed to compute this function on *each and every input* of sufficiently large length.

1.1 High-end results: Breaking the $\mathcal{PSPACE}$ barrier

Our first main result is the following uniform hardness to randomness tradeoff. Consider the class of logspace-uniform circuits[6] of exponential size $2^{O(n)}$ and near-exponential depth $2^{o(n)}$. Observe that this class contains $\mathcal{SPACE}[O(n)]$ and in fact seems much broader than it: Indeed, $\mathcal{SPACE}[O(n)]$ can be simulated even by logspace-uniform circuits of size $2^{O(n)}$ and smaller depth $\mathrm{poly}(n)$ rather than $2^{o(n)}$ (by the standard approach of repeated squaring). We prove that if the foregoing class contains a function hard for probabilistic algorithms with running time $2^{\epsilon \cdot n}$, then $\mathcal{RP}$ and $\mathcal{BPP}$ can be derandomized in polynomial time on average, as follows:

Theorem 1.1 (high-end hardness vs randomness beyond $\mathcal{PSPACE}$)**.** *For every $\epsilon > 0$ there exists $\delta > 0$ such that the following holds. Assume that there is a function $L \subseteq \{0,1\}^*$ computable by logspace-uniform circuits of size $2^{O(n)}$ and depth $2^{\delta \cdot n}$ such that $L \notin \mathtt{i.o.}\mathcal{BPTIME}[2^{\epsilon \cdot n}]$. Then, for every $a \in \mathbb{N}$ it holds that*

$$\mathcal{RP} \subseteq \mathsf{heur}_{1-1/n^a}\text{-}\mathcal{P} \ ,$$
$$\mathcal{BPP} \subseteq \mathsf{heur}_{1-1/n^a}\text{-}\mathcal{P}/O(\log n) \ .$$

The meaning of "heur_{1-1/n^a}" above is that for every $L \in \mathcal{RP}$ and *every polynomial-time samplable distribution* $\mathbf{x}$ there exists $L' \in \mathcal{P}$ such that $\Pr_{x \sim \mathbf{x}}[L(x) = L'(x)] \geq 1 - 1/|x|^a$, and ditto for $\mathcal{BPP}$ and $\mathcal{P}/O(\log(n))$ (see Definition 3.4). When the depth of the circuits for the hard function is smaller, say $\mathrm{poly}(n)$, the advice for derandomizing $\mathcal{BPP}$ is shorter, say $O(\mathrm{loglog}(n))$ (see Theorem 5.2 for precise details).

We stress that there are several novel features in Theorem 1.1. First, it relies on hardness for functions that are (conjectured to be) *outside of* $\mathcal{PSPACE}$; in particular, these functions are not necessarily downward self-reducible. Secondly, it relies on hardness for an *arbitrary* function in the class, rather than only for specific functions with useful structure. Thirdly, the tradeoff is smooth, and in particular applies to the "high-end" regime of parameters (when hardness is $2^{\epsilon \cdot n}$ and the derandomization is in polynomial time); our result is indeed more general, covering the entire parameter range (see Theorem 5.2). And as a fourth point, the derandomization algorithm works on *all input lengths*, rather than only on infinitely many input lengths.

Optimal worst-case to average-case reduction. A salient feature of Theorem 1.1 is that we assume *worst-case hardness* and yet deduce derandomization that succeeds on $1 - o(1)$ of the inputs. One might suspect that the proof will go through a worst-case to average-case reduction for probabilistic algorithms (i.e., a reduction of computing a function in the worst-case to computing it on $o(1)$ of the inputs). In fact, the reduction that seems to be implicit in the result should be essentially optimal, since the conclusion of Theorem 1.1 does not have super-polynomial overheads in the algorithm's running time.

Prior to our work, *optimal* worst-case to average-case reductions for probabilistic algorithms were known either for $\mathcal{E}$ (see [TV07]) or for small subclasses of $\mathcal{P}$ (see, e.g., [GR17]). However, for classes such as the one in Theorem 1.1, the known reductions relied on hardness only for specific problems, and moreover these problems were computable in time $2^{o(n)}$ (and thus cannot have hardness $2^{\Omega(n)}$; see [TV07] for details).

[6] Recall that a circuit family of size $s(n)$ is logspace-uniform if there is a machine that gets input 1^n, uses $O(\log(s(n)))$ space, and prints the n^{th} circuit in the family (see Definition 3.5).

On the way to proving Theorem 1.1 we are indeed able to prove an optimal worst-case to average-case reduction for computing functions in complexity classes such as the one in Theorem 1.1. We now state what seems to us as the most interesting special case, which is an optimal worst-case to average-case reduction for computing functions in $\mathcal{SPACE}[O(n)]$ by probabilistic algorithms.

Theorem 1.2 (optimal worst-case to average-case reduction for linear space; informal, see Theorem 5.4)**.** *For every "nice" $\epsilon(n)$ and $T(n)$, if $\mathcal{SPACE}[O(n)] \not\subseteq \texttt{i.o.}\mathcal{BPTIME}[T]$, then $\mathcal{SPACE}[O(n)]$ is hard to compute on more than $(1/2+\epsilon)$ of the inputs in probabilistic time $T(n/c)\cdot(\epsilon/n)^c$, for a constant $c>1$, on all sufficiently large input lengths $n\in\mathbb{N}$.*

As a corollary of Theorem 1.2, if $\mathcal{SPACE}[O(n)] \not\subseteq \texttt{i.o.}\mathcal{BPTIME}[2^{\delta\cdot n}]$, then $\mathcal{SPACE}[O(n)]$ cannot be successfully computed on $1/2+2^{-\delta'\cdot n}$ of the inputs in probabilistic time $2^{\delta'\cdot n}$, where $\delta' = \Theta(\delta)$. The main technical result underlying Theorem 1.2 is a construction of a new instance-checkable problem that is complete for $\mathcal{SPACE}[O(n)]$ under linear-time reductions (see Section 2 for details).

1.2 Fine-grained hardness for unstructured problems

As mentioned above, a second type of uniform hardness vs randomness results focuses on *fine-grained hardness*; that is, showing average-case derandomization under assumptions that functions in $\mathcal{P}$ cannot be solved in some fixed polynomial time.

Results of this type that rely on hardness for *non-uniform circuits* have been extensively studied. Specifically, following Goldreich and Wigderson [GW02], a sequence of works culminated in the following result by Kinne, van Melkebeek, and Shaltiel [KMS12] (see also [MS05; Sha11; Sha10]): If for every k there is $L_k \in \mathcal{P}$ that is hard to compute with less than $1/n$ errors by non-uniform circuits of size n^k (for all $n\in\mathbb{N}$), then $\mathcal{BPP}$ can be derandomized in polynomial time on average (over the uniform distribution, with error $1/\mathrm{poly}(n)$; see [KMS12, Theorem 1]).

Since the conclusion is an average-case derandomization, a natural goal is to try and relax the hypothesis and only assume hardness for *uniform* probabilistic algorithms (rather than for circuits). Recently, Carmosino, Impagliazzo and Sabin [CIS18] showed the first result along these lines: They deduced average-case derandomization from hardness of *specific problems in $\mathcal{P}$*, namely of counting k-cliques or for k-orthogonal-vectors. Indeed, the latter problems have a structure similar to the one required in classical results, namely they are downward self-reducible in some sense (see [CIS18, Section 2.1], following [BRS+17]). Nevertheless, their work managed to bypass some of the traditional obstacles (e.g., getting derandomization in polynomial time or on all input lengths, similarly to what we were able to obtain in Section 1.1).

In this context too, our goal is to get rid of the structural requirements and of the dependency on hardness of specific problems, while simultaneously significantly improving on the parameters. Our first result starts from mild average-case hardness for *any function in a large natural subclass of $\mathcal{P}$*: Namely, the class of problems that can be decided by logspace-uniform circuits of polynomial size and fixed polynomial depth, say n^3. (Indeed, note that this upper bound refers to uniform circuits of *polynomial* depth rather than only to logspace-uniform $\mathcal{NC}$.) That is:

Theorem 1.3 (derandomization from mild average-case fine-grained hardness)**.** *Fix $d\in\mathbb{N}$, and assume that for every $c\in\mathbb{N}$ there is a problem $L\subseteq\{0,1\}^*$ computable by logspace-uniform circuits of polynomial size $n^{O_c(1)}$ and depth n^d such that $L\notin \texttt{i.o.-avg}_{(1-n^{-d})}\text{-}\mathcal{BPTIME}[n^c]$.*

Then, for every $a \in \mathbb{N}$ it holds that

$$\mathcal{RP} \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P} \ ,$$

where the notation avg *refers to average-case simulation over the uniform distribution.*[7]

Indeed, Theorem 1.3 gets very close to achieving the goal of simply replacing the non-uniform hardness assumption in the result of [KMS12] by a uniform hardness assumption; the only remaining gaps are that we require a fixed polynomial depth upper bound and that we derandomize $\mathcal{RP}$ rather than $\mathcal{BPP}$. (This is indeed reminiscent of the gaps between Theorem 1.1 and the "ideal" result mentioned there.)

The specific problems considered prior to our work (i.e., k-clique and k-orthogonal-vectors) belong to the class in Theorem 1.3, and in fact also to a smaller subclass that will be considered next. We stress that the hypothesis in Theorem 1.3 only assumes that every n^c-time algorithm fails on a n^{-a}-fraction of the inputs (i.e., we assume a mild average-case hardness), but the conclusion is that the derandomization succeeds on the vast majority of inputs (i.e., on a $1 - n^{-d}$ fraction).

Derandomization from worst-case fine-grained assumptions. While the average-case hardness assumption in Theorem 1.3 is quite mild, it is still stronger than a worst-case hardness assumption. In the following result we strike a different tradeoff. We define a natural subclass of $\mathcal{P}$ (we encourage the reader to intuitively think of it as a subclass of the one in Theorem 1.3) that consists of functions computable by logspace-uniform *arithmetic formulas of arbitrary polynomial size and fixed polynomial degree*; for example, arithmetic formulas of size $\mathrm{poly}(n)$ and degree n^2. That is:

Definition 1.4 (low-degree arithmetic formulas)**.** *Let $d \in \mathbb{N}$, let $p(n)$ be a function mapping integers to prime powers such that $n^4 \leq p(n) \leq \mathrm{poly}(n)$, and let $g = \{g_n\}$ such that $g_n \colon [p(n)] \to \{0,1\}^*$ is computable in space $O(\log(n))$. Let $F = \{F_n\}$ be a family of logspace-uniform arithmetic formulas of degree n^2 and polynomial size over $\mathbb{F}_{p(n)}$. Consider the problem $\Pi = \Pi^{F,p,g}$ in which the input is x and the output is $g(F(x))$.*

Assuming that this class is hard, in the *worst-case*, for probabilistic algorithms running in any fixed polynomial time n^c, we deduce that $\mathcal{RP} = \mathcal{P}$ on average:

Theorem 1.5 (derandomization from worst-case fine-grained hardness for low-degree arithmetic formulas)**.** *Assuming that for every $c \in \mathbb{N}$ there are some g and F and p (as in Definition 1.4) such that $\Pi^{F,p,g} \notin \mathtt{i.o.}\mathcal{BPTIME}[n^c]$. Then for every $a \in \mathbb{N}$*

$$\mathcal{RP} \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P} \ .$$

Intuitively, one should think of Theorem 1.5 as starting from hardness in a smaller subclass than that of Theorem 1.3, but requiring only worst-case hardness rather than mild average-case hardness. (The reason that we intuitively think of the class in Theorem 1.5 as a subclass of the one in Theorem 1.3 is that formulas of fixed polynomial degree can be evaluated in small depth; see Section 2 for details.)

[7] That is, the notation "$\mathcal{C} \in \mathsf{avg}_{(1-\delta)}\text{-}\mathcal{C}'$" means that for every $L \in \mathcal{C}$ there exists $L \in \mathcal{C}'$ such that for every sufficiently large $n \in \mathbb{N}$ it holds that $\Pr_{x \in \{0,1\}^n}[L(x) = L'(x)] \geq 1 - \delta$.

A comparison of the parameters to previous work. As mentioned above, beyond the fact that we start from hardness of arbitrary functions in natural subclasses of $\mathcal{P}$, our results also significantly improve on the *parameters* of previous work. To see this, recall that [CIS18] proved that if counting k-cliques in a given n-vertex graph requires probabilistic time $n^{(1/2+\epsilon)\cdot k}$ for some $\epsilon > 0$, then $\mathcal{BPP} = \mathcal{P}$ on average over the uniform distribution. Instantiating Theorem 1.5 for the special case of this problem (indeed, one can count k-cliques with low-degree arithmetic formulas as in Definition 1.4), we obtain the following corollary:

Corollary 1.6 (derandomization from hardness of k-clique, for comparison)**.** *Assume that for every $c \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that counting k-cliques is hard for probabilistic time n^c on all input lengths. Then, for every $a \in \mathbb{N}$*

$$\mathcal{RP} \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P} \ .$$

The difference in hypotheses between Corollary 1.6 and the result of [CIS18] is that the latter requires the hardness $n^{h(k)}$ of k-clique to grow as $h(k) = (1/2+\epsilon)\cdot k$, whereas we only require that $h(k)$ will be an unbounded function. As a consequence of our improved parameters, our results also immediately imply an affirmative answer to the main open problem in [CIS18], which asked to obtain similar results for problems such as k-SUM that can be solved in time $O(n^{\lceil k/2 \rceil})$.

As demonstrated by the special case of Corollary 1.6, the assumptions in Theorems 1.3 and 1.5 are arguably among the most believable assumptions that are currently known to imply polynomial-time derandomization. Indeed, the only caveat is that we derandomize $\mathcal{RP}$ rather than $\mathcal{BPP}$ (see Section 2.4 for an explanation why).

2 Technical overview

The technical starting-point for our work is a non-black-box derandomization algorithm from [CT21], called a reconstructive targeted HSG. This algorithm H_f relies on a hard function f that is computable by logspace-uniform circuits of size T and bounded depth $d \ll T$ to solve the following task: The algorithm *gets input* $x \in \{0,1\}^n$, and prints a set $H_f(x)$ of n-bit strings that is, hopefully, pseudorandom for every efficient algorithm *that also gets access to the same input* x.

The analysis of this algorithm works via a reconstruction argument: Any efficient algorithm that gets input x and distinguishes (the uniform distribution on) $H_f(x)$ from uniformly random strings can be converted into an algorithm that computes f quickly *at the same input* x.[8] Thus, the hardness of f is converted into randomness "instance-wise", for every fixed input. Indeed, a caveat here is that pseudorandomness is only guaranteed for probabilistic algorithms with one-sided error – the reconstruction relies on the assumption that $A_x(\cdot)$ accepts a uniformly random string, with high probability, but rejects all strings in $H_f(x)$. (See Theorem 4.5 for precise details.)

A recurring challenge: Worst-case to average-case reductions. At a high-level, in this work we start with worst-case hardness assumptions (or with mild average-case hardness assumptions); that is, we assume that every algorithm fails to compute f at *one input* (or on a small fraction of inputs). However, since H_f translates hardness

[8] Indeed, more formally, for every efficient algorithm A there exists an efficient algorithm F such that for every fixed x, if $A(x,\cdot) = A_x(\cdot)$ is a distinguisher for $H_f(x)$ then $F(x) = f(x)$.

into randomness "instance-wise", if we want to use H_f to obtain derandomization that succeeds on $1-o(1)$ of inputs, we need a function that is hard on $1-o(1)$ of the inputs. Thus, many of our results will include worst-case to average-case reductions, which imply that if a function f as above is hard on the worst-case, then there is another function f' with similar complexity that is hard on $1-o(1)$ of the inputs. (Other results will include reductions of computing a function successfully on $1-o(1)$ of the inputs to computing it on $o(1)$ of the inputs.)

2.1 Proofs of Theorems 1.1 and 1.2

The first technical result in our work is a construction of a new instance-checkable problem. Recall that a problem L is instance-checkable if there is a probabilistic algorithm M that gets input x and oracle $\tilde{L}$, and with high probability, if $\tilde{L} = L$ then $M(x) = L(x)$, and for any $\tilde{L}$ satisfies $M(x) \in \{L(x), \perp\}$ (see Definition 3.11). An instance checker is useful for reductions of computing f in the worst-case to computing some $\tilde{f}$ in the average-case: This is because such reductions usually rely on local list-decoding of error-correcting codes to produce a *list of candidate procedures* for f, and an instance checker allows us to test each candidate and only "trust" the answer of ones who are correct (see, e.g., [TV07, Section 5] for further explanation).

The basic version of our instance checker. For every logspace-uniform circuit C of size $T(n) \leq 2^{O(n)}$ and depth $d \ll T$, we construct a problem $L_C \subseteq \{0,1\}^*$ such that:

1. Computing C reduces in linear time to computing L_C.
2. L_C has approximately the same complexity as C.
3. L_C has an instance checker that runs in time $\text{poly}(n, d, \log(T))$ and given x only makes queries of length $|x|$.

Crucially, since the reduction runs in linear time, if C is hard for probabilistic algorithms running in time $T(n)$, then L_C is also hard for probabilistic algorithms running in similar time $T(\Omega(n))$. And since we can construct L_C for any C of complexity as above, it means that if *any* such C is hard for time $T(n)$, then there is an instance-checkable problem L_C with similar hardness $T(\Omega(n))$.

The construction of $L = L_C$ is based on ideas from the doubly efficient interactive proof system of Goldwasser, Kalai, and Rothblum [GKR15]. Loosely speaking, for any logspace-uniform circuit family C of size T and depth d and any input $x \in \{0,1\}^n$, they showed a way to encode the computation of $C(x)$ as a matrix M_x whose entries are in a field $\mathbb{F}$ of size $\text{poly}(T)$ such that the following holds: Verifying a claimed value for the $(i,j)^{th}$ entry in M_x reduces in probabilistic time $\text{poly}(n, d, \log(T))$, and via additional queries to M_x, to a predicate on the input x that is also computable in time $\text{poly}(n, \log(T))$.

The main idea in our construction of L is to define its inputs as (x, i, j, k), where x is an input to C and (i,j) is an index in M_x (and $k \in [\log(|\mathbb{F}|)]$ is the index of a bit in the representation of $\mathbb{F}$-elements). The instance checker simulates the verifier of [GKR15], reducing the computation of L at any given (x, i, j, k) to verification of L at other points corresponding to M_x, and then finally to an efficient computation on the input x. Since the matrix M_x is of size $\text{poly}(T) \leq 2^{O(n)}$, the length of an index (i,j) is at most $O(n)$, and thus the blow-up in input length from inputs for C to inputs

for L is only linear.[9] And indeed, the encoding of $C(x)$ into the matrix M_x is not computationally expensive, which means that the complexity of L_C is not much larger than that of C;[10] see Proposition 4.4 for a precise statement and a proof.

Proof of Theorem 1.1. The proof of Theorem 1.1 will use the instance checkable L above, but it does not explicitly rely on a worst-case to average-case reduction. Assume that some C of size $2^{O(n)}$ and depth $d = 2^{o(n)}$ computes a function that is hard for $\mathcal{BPTIME}[2^{\epsilon \cdot n}]$, and let $L = L_C$ be the problem above. The main idea in the proof is to apply the generator H_f to the function f that maps any input $x \in \{0,1\}^n$ to the *truth-table of L on $\ell = O(\log(n))$ input bits*; that is, the hard function $f\colon \{0,1\}^n \to \{0,1\}^{2^\ell}$ prints the entire truth-table of L_ℓ (where L_ℓ denotes the restriction of L to inputs of size ℓ). Since L_ℓ is computable by logspace-uniform circuits with similar complexity to that of C, we can also compute f with a circuit of approximately the same complexity (by computing the output bits in parallel).

Now, to simulate a probabilistic linear-time algorithm A on input $x \in \{0,1\}^n$, we compute $H_f(x)$ and output $\vee_{s \in H_f(x)} A(x,s)$. [11] Why does this derandomization work, on average, over any polynomial-time samplable distribution? Assume that an efficient sampling algorithm S succeeds, with probability $1/n$, in finding x such that $\Pr_r[M(x,r) = 1] \geq 1/2$ but $M(x,s) = 0$ for every $s \in H_f(x)$. (For simplicity let us assume that S runs in linear time too.) For any such x, the reconstruction algorithm R for $H_f(x)$ asserts that in this case we can compute $f(x)$ in time $|f(x)| \cdot n^c$, where n^c is much smaller than the hardness $2^{\epsilon \cdot \ell}$ of L_ℓ.

We would like to use this to contradict the worst-case hardness of L_ℓ. There are two problems, however. First, the output size of f is much larger than the hardness of L_ℓ (i.e., $|f(x)| = 2^{O(\ell)} \gg 2^{\epsilon \cdot \ell}$), making the reconstruction R too inefficient to yield a contradiction. To handle this problem, we observe that the reconstruction algorithm of H_f satisfies a stronger property: Not only can it print $f(x)$ in time $|f(x)| \cdot n^c$, it can actually print a circuit $C_{f(x)}$ of size n^c whose truth-table is $f(x)$ (see Theorem 4.5).[12] Thus, we can compute L_ℓ at any input $q \in \{0,1\}^\ell$ (i.e., compute the q^{th} bit of $f(x)$) by running R to obtain $C_{f(x)}$ and outputting $C_{f(x)}(q)$.

The second problem is that the procedure above succeeds only with low probability $1/n$ (i.e., the probability that S finds an x such that $M(x,\cdot)$ is a distinguisher). We overcome this using the instance checker: Given input $z \in \{0,1\}^\ell$ we run S for $k = O(n)$ times obtaining $x_1, ..., x_k$, and for each $i \in [k]$ we run the instance checker with input z, while answering each of its queries $q \in \{0,1\}^\ell$ with the reconstruction $R(q)$ and the distinguisher $D_{x_i}(r) = M(x_i, r)$. Assuming that at least one x is "good", and that all invocations of the instance checker and of R were correct, the instance checker will output $L_\ell(z)$ for some $i \in [k]$, and will either output $\perp$ or $L_\ell(z)$ for all $i \in [k]$, allowing us to deduce $L_\ell(z)$. The running time of this procedure is some

[9] In fact, the blow-up is additive $n \mapsto n + O(\log(T))$, where $T \leq 2^{O(n)}$.

[10] Loosely speaking, the encoding M_x of $C(x)$ involves arithmetizing each layer of $C(x)$ via a low-degree extension, and adding a small number of intermediary low-degree polynomials between each pair of layers. Both the low-degree extensions and the intermediary low-degree polynomials can be efficiently computed from the original layers of $C(x)$.

[11] In the overview we focus on derandomizing $\mathcal{RTIME}[O(n)]$, for simplicity.

[12] This is the case because the reconstruction argument iteratively reconstructs circuits of small size (i.e., less than n^c) for each of the $2^{o(n)}$ layers of the circuit for f, starting from the bottom (input) layer and working its way up to the top (output) layer. Thus, in the last step it obtains a circuit whose truth-table is (a low-degree extension of) the string $f(x)$. See the proof of Theorem 4.5 for precise details.

fixed polynomial, we can ensure that it is less than $2^{\epsilon \cdot \ell}$ by taking $\ell = O(\log(n))$ to be sufficiently large.

Derandomization of $\mathcal{BPP}$. The foregoing argument yields derandomization of $\mathcal{RP}$. To deduce derandomization of $\mathcal{BPP}$ with short advice, we observe that the targeted generator H_f is not only a targeted hitting-set generator, but also a *targeted somewhere-PRG*; that is, it outputs a collection of $d'(\ell) \approx d(\ell) = n^{o(1)}$ lists $W_1, ..., W_{d'}$ of strings, and for every efficient algorithm D there exists $i \in [d']$ such that W_i is pseudorandom for D, where pseudorandomness here is in the usual sense of *two-sided error*.

We want to use this targeted somewhere-PRG to argue that for every machine M and sampling algorithm S there exists $i \in [d']$ such that the probability that S samples an input x for which $M(x, \cdot)$ is a distinguisher for W_i is at most $1/n$. Given this claim, we can hard-wire i into the derandomization algorithm as advice of length $\log(d')$, and the derandomization algorithm will only use the pseudorandom strings in W_i.

To show the claim above, assume the opposite: For each $i \in [d']$, with probability at least $1/n$ the sampling algorithm S outputs x such that $M(x, \cdot)$ is a distinguisher for W_i. Recall that the pseudorandomness of the generator was established by a reconstruction argument, asserting that a distinguisher D can be used to compute L_ℓ too quickly. We show a *stronger reconstruction procedure*, which works not only when it is given a distinguisher D, but also when it is given a sequence of d' sets of functions such that for $i \in [d']$, the i^{th} set contains a distinguisher for W_i. (Intuitively, this reconstruction procedure implicitly performs iterative "instance-checking": It works in d' iterations, and in each iteration it is able to find the "good" distinguisher among the candidate functions in the corresponding set.) For each $i \in [d']$, we call S for $O(n \cdot \log(d'))$ times to sample a set X_i of inputs, such that with high probability, for every $i \in [d']$ there is $x_i \in X_i$ such that $D_{x_i}(\cdot) = M(x_i, \cdot)$ is a distinguisher for W_i. This satisfies the hypothesis of the stronger reconstruction procedure, allowing us to contradict the hardness of L_ℓ. For precise details see the proofs of Theorems 4.5 and 5.3.

Proof of Theorem 1.2. We want to prove an optimal worst-case to average-case reduction for computing $\mathcal{SPACE}[O(n)]$ by probabilistic algorithms, and the main challenge will be to refine the instance checker above. At a high-level, our reduction follows a standard plan: Given $L^{(0)} \in \mathcal{SPACE}[O(n)]$ that is hard for probabilistic algorithms in the worst case, we reduce $L^{(0)}$ to an instance-checkable L, encode the truth-table of L by a locally list-decodable code Enc that is computable in space $O(n)$ (see Theorem 3.8), and the reduction applies the instance checker with each of the candidate circuits that the local decoder outputs (we do not elaborate on this, since the general approach is well-known; see the proof of Theorem 1.2 for details).

The challenge is that L above is "complete" for logspace-uniform circuits of size T and depth $d \ll T$,[13] whereas *we want L to be complete for $\mathcal{SPACE}[O(n)]$* (both notions of completeness here refer to linear-time reductions). Indeed, any function $L^{(0)} \in \mathcal{SPACE}[O(n)]$ has circuits of size $2^{O(n)}$ and depth $\mathrm{poly}(n)$, using the standard technique of repeatedly squaring the transition matrix of the linear-space machine M for $L^{(0)}$, and moreover these circuits are logspace-uniform. The crucial observation is that given an input $x \in \{0,1\}^n$ and an index of a gate $g \in [2^{O(n)}]$ in this circuit, *we can compute in linear space the value of $g(x)$*. This is because every gate g is associated with two instantaneous configuration γ, γ' of M, and $g(x)$ indicates whether or not running

[13] We write "complete" because the circuit for L_C is somewhat larger and deeper than the circuit C.

M for $i \leq 2^{O(n)}$ steps, starting from the configuration γ, results in the configuration γ'. Thus, to compute $g(x)$ we can simply simulate M starting from configuration γ, and check whether its configuration after i steps is γ'.

Given this property, we observe that all the steps required to compute L (i.e., to compute an entry in M_x) maintain the linear-space complexity. Intuitively, this is because these steps mainly involve computing low-degree extensions of the layers of the circuit for $L^{(0)}$ (or simple reductions between a constant number of low-degree extensions), and these can be carried out in space $O(n)$ with oracle access to the gates of the original circuit. See Proposition 4.3 for further details.

2.2 Proof of Theorem 1.3

At a high-level, the plan for proving Theorem 1.3 is as follows. We assume that there is a problem $L^{(0)}$ computable by logspace-uniform circuits of polynomial size and depth n^2 such that $L^{(0)} \notin \mathsf{avg}_{(1-1/n)}\text{-}\mathcal{BPTIME}[n^c]$.[14] Since $L^{(0)}$ is reducible in linear time to the instance-checkable problem L described in the beginning of Section 2.1, we hope to prove that L will also have essentially the same hardness. We then encode L via the k-wise direct product code with $k = \tilde{O}(n^2)$ repetitions, to obtain a problem $L^{\otimes k}$ with essentially the same computational complexity,[15] and use the instance checker as well as the celebrated direct product theorem of Impagliazzo *et al.* [IJK+10] to argue that $L^{\otimes k}$ cannot be computed in fixed polynomial time even on (say) $1/n^3$ of the inputs (see below). Finally, we use $L^{\otimes k}$ as the hard function for the targeted HSG H_f, obtaining derandomization that runs in polynomial time and succeeds on $1 - 1/n^3$ of the inputs.

There are two parts in the plan above that we left vague: The claim that L is mildly hard on average (supposedly, because of the reduction from $L^{(0)}$, which is mildly hard on average); and the claim that $L^{\otimes k}$ is hard on $1 - 1/n^3$ of inputs (supposedly, because it is a k-wise direct product of L). The challenges that underlie the proofs of both claims are similar, so for simplicity we focus on the claim that $L^{\otimes k}$ cannot be computed in time close to n^c even on $1/n^3$ of the inputs.

For a large enough $k = \tilde{O}(n^3)$, assuming towards a contradiction that $L^{\otimes k}$ can be computed on at least $1/n^3$ of the inputs in time $n^{c'}$, we want to contradict the hardness of L. Recall that [IJK+10] yields a list-decoder that, with probability $\Omega(n^{-3})$, outputs a circuit of size $\mathrm{poly}(n^{c'})$ that computes L correctly on $1 - 1/n^2$ of the inputs. Given an input (x, i, j) for L,[16] we can repeatedly invoke the list-decoder to obtain a list of $t = O(n^3)$ circuits $C_1, ..., C_t$, and run the instance checker with each C_i, hoping to be "convinced" by the good C_i and not misled by all other C_i's.

The gap in the foregoing argument is that C_i only computes L correctly on $1 - n^{-2}$ of the inputs rather than on all inputs, and our instance checker is not guaranteed to work with such C_i's. The reason is that, in contrast to what one might expect when thinking of instance checkers, the queries of our instance checker are *not* uniform. (Indeed, one can design an adversarial C_i that fails this instance checker.)

Tolerant instance checkers. To bridge the foregoing gap we modify the instance checkable problem to a problem whose instance checker is more resilient. Specifically, we introduce the notion of tolerant instance checkers, which are instance checkers

[14] We use convenient parameters in the current section, for simplicity.

[15] Recall that the k-wise direct-product of L takes input $\bar{x} = (x_1, ..., x_n) \in (\{0,1\}^n)^k$ and outputs the k bits $L(x_1), ..., L(x_k)$. In particular, we can compute the k output bits in parallel.

[16] In this section, for simplicity of presentation, we ignore the fourth component in inputs to L, whose only function is to transform L into a Boolean function.

that, when given an oracle that agrees with the target problem L on $1-\epsilon$ of the inputs, satisfy the completeness requirement of a standard instance checker on at least $1-\epsilon'$ of the inputs, for $\epsilon' \approx \epsilon$ (see Definition 3.12).

We then refine the instance checkable problem L above so that it indeed has a *tolerant* instance checker, rather than only a standard one. Specifically, recall that the matrix M_x in the definition of L consists of $d' = \tilde{O}(d)$ rows where each row is a low-degree polynomial $\mathbb{F}^m \to \mathbb{F}$ (for a suitable choice of $m \in \mathbb{N}$), and in entry (i,j) we have the evaluation of the polynomial $\hat{\alpha}_i$ at the input indexed by j, denoted $\vec{j} \in \mathbb{F}^m$. For every fixed x we define a polynomial $p_x \colon \mathbb{F} \times \mathbb{F}^m \mapsto \mathbb{F}$ that interpolates all the d' polynomials; that is, when p_x gets as input (i,j) where $i \in [d']$ it outputs $\hat{\alpha}_i(\vec{j})$, and otherwise (when $i \notin [d']$) it outputs an interpolation of the d' polynomials. Since the number d' of polynomials is sufficiently small, the polynomial p_x is of low degree.

Now, we modify the definition of L such that it gets input (x,i,j) where $i \in \mathbb{F}$ may also be outside $[d']$, and we prove that this new version has logspace-uniform circuits with essentially the same depth as the previous version and with only a polynomial size overhead, and that also has an instance checker with the same time complexity as the previous version. The reason that these two claims hold is that d' is small (given that we start from $L^{(0)}$ whose circuits have fixed polynomial depth but larger polynomial size), and hence to compute L we just need to interpolate a small number of polynomials (see Proposition 4.4 for details). We obtain the following two properties:

1. Given input (x,i,j), the instance checker only makes queries of the form (x,i',j'); that is, all queries have the same first component x as the input.[17]

2. For every fixed x, the function $p_x(i,j) = L(x,i,j)$ is a low-degree polynomial.

To see that this problem has a tolerant instance checker, note that if an oracle agrees with L on most inputs (x,i,j), then for most x it agrees with p_x with high probability over (i,j), say $9/10$. Thus, for most x the instance checker can use *self-correction of the low-degree polynomial* p_x, and run the original instance checker while simulating access to the actual polynomial p_x (again, see Proposition 4.4 for precise details).

Using the refined instance checker to bridge the gaps. Let us see how we use these properties to bridge the gaps in our proof. Recall that in our "towards a contradiction" argument (when we assumed that $L^{\otimes k}$ was "too easy"), when repeating the list-decoder we obtained a list of circuits $C_1, ..., C_t$, and at least one C_w computes L on $1-n^{-2}$ of the inputs. We can thus run the tolerant instance checker with each of these circuits C_i as oracle: The soundness condition holds on every input and with each oracle, whereas the tolerant completeness condition guarantees that there is a set of approximately $1-n^{-2}$ inputs such that when the instance checker uses oracle C_w it is able to compute L correctly. This yields the contradiction that we wanted.

(The proof is actually a bit more cumbersome technically, since we want to preserve hardness on almost all input lengths. This requires us to also use a tolerant instance checker for $L^{\otimes k}$, which tolerates very high corruption; such a tolerant instance checker can be obtained directly from the tolerant instance checker for L. For details see Claim 3.12.1 and Lemma A.5 in Appendix A.)

[17] Indeed, our previous construction of the instance checker already has this property, and it is maintained when interpolating the polynomials into p_x; see the proof of Proposition 4.4. Also, for simplicity of presentation we ignore the additional input k that converts L into a Boolean function.

Strongly tolerant instance checkers. A similar argument allows us to prove that L is mildly hard on average, based on the mild average-case hardness of $L^{(0)}$. However, since we are now trying to preserve very mild hardness on all input lengths under reductions, the argument turns out to be more subtle, and requires us to introduce a more refined notion of strongly tolerant instance checkers. The instance checker presented above is already strongly tolerant, and using it the argument carries through. For technical details see Definition 3.13 and Lemma A.2 in Appendix A.

2.3 Proof of Theorem 1.5

Recall that we now want to prove derandomization assuming *worst-case* hardness of a function computable by low-degree arithmetic formulas of polynomial size. The intuition underlying the proof of Theorem 1.5 is that arithmetic formulas can be *balanced* to be of logarithmic depth, by a very efficient algorithm; hence, this class of formulas is essentially a subclass of the one from Theorem 1.3. Moreover, since the formulas have low-degree, this class supports a worst-case to mild average-case reduction.

Thus, our goal is to start from worst-case hardness for our class of arithmetic formulas, argue that the formulas can be balanced while maintaining their complexity, deduce mild average-case hardness, and then invoke Theorem 1.3 as a black-box.

Balancing the formula by low-depth circuits. For any logspace-uniform arithmetic formula F_n of degree n^2, we show that the corresponding polynomial P_n can be computed in logspace-uniform $\mathcal{NC}$ (i.e., the circuit computing P_n has depth $\text{polylog}(n)$).

By a standard argument (see, e.g., [SY10, Theorem 2.6]), any arithmetic formula of polynomial size can be converted into an equivalent arithmetic circuit of polynomial size and depth $O(\log(n))$. Our key observation is that this balancing algorithm is quite simple: In particular, the bottlenecks of the procedure are finding a "center of mass" of a binary tree,[18] and computing a certain partial derivative, both of which can be done in logspace-uniform $\mathcal{NC}$. With this observation in mind, the "balancing" procedure can be carried out in $O(\log(n))$ stages, with each stage implementable in logspace-uniform $\mathcal{NC}$. After the balancing, we evaluate the $O(\log(n))$ depth arithmetic circuit in logspace-uniform $\mathcal{NC}$ to compute P_n (see Lemma 7.3.2 and Section 7.1.1 for details).[19]

Technical complications when working with prime fields. In some settings we will need to consider the formula as a polynomial over a *large prime field*; this happens, for example, when considering arithmetic formulas for counting problems (such as counting k-cliques). A standard complication in this setting is that the average-case complexity of the problem is sensitive to the Boolean encoding of field elements (see the proof of Lemma 7.3 for details). An additional complication in this setting is that in the worst-case to average-case reduction, we need to deterministically and quickly find such a prime (e.g., find a prime of size n^{100} in deterministic time n^2), but such an algorithm isn't known. Thus, in our worst-case to average-case reduction we actually define an auxiliary problem in which the prime is incorporated into the truth-table. (See the proof of Lemma 7.3 for a careful implementation of this idea.)

[18] Given a binary tree (meaning that each node has at most two children) T of n nodes, a node u is called a "center of mass", if the size of the sub-tree rooted at u has size between $[n/3, 2n/3]$.

[19] We remind the reader that arithmetic *circuits* can also be balanced, albeit by a more complicated algorithm (see [VSB+83]). We did not try to extend our results to hold for this model.

2.4 Why only $\mathcal{RP}$?

Let us explain the technical challenge due to which we were only able to derandomize $\mathcal{RP}$ in Theorems 1.3 and 1.5, rather than $\mathcal{BPP}$. The same technical challenge also existed in [CT21], and in fact it dates back at least 25 years, to the work of Impagliazzo and Wigderson [IW98] that founded the area of uniform hardness vs randomness.

Fix a uniform probabilistic linear-time machine M whose coins we wish to replace by pseudorandom coins on a given input x. Assume that we can produce, in time $\text{poly}(n)$, a sequence of n sets $S_1, ..., S_n \subseteq \{0,1\}^n$, each consisting of $\text{poly}(n)$ strings, and we are guaranteed that for every x there *exists* $i \in [n]$ such that S_i is pseudorandom for M with input x. Can we *combine* the n sets, perhaps using additional $O(\log(n))$ random bits, into a single set S that is guaranteed to be pseudorandom for M with x?

Indeed, this challenge refers to the *computational version* of an object known in extractor theory as *mergers*; it is thus apt to refer to it as considering computational mergers. While we know how to construct computational mergers in other setting – for example, when the distinguisher class is non-uniform – in our setting where M is uniform (and does not have enough time to compute the strings in the S_i's by itself), we do not know how to solve this. This obstacle prevented many previous works from obtaining average-case derandomization on all input lengths (see, e.g., [IW98; CNS99; TV07; CRT+20]), and significantly increased the running time of the worst-case derandomization in [CT21] when it was scaled to the "low-end" parameter setting.

3 Preliminaries

For two real numbers v and $\epsilon > 0$, we use the notation $v \pm \epsilon$ to denote the interval $[v - \epsilon, v + \epsilon]$. Throughout the paper we will denote random variables in boldface fonts. We will denote the uniform distribution over $\{0,1\}^n$ by $\mathbf{u}_n$, and the uniform distribution over a set $[n]$ by $\mathbf{u}_{[n]}$. Recall the standard definition of a distinguisher for a distribution:

Definition 3.1 (distinguisher)**.** *We say that $D\colon \{0,1\}^n \to \{0,1\}$ is an ϵ-*distinguisher *for a distribution $\mathbf{x}_n$ over $\{0,1\}^n$ if $\Pr[D(\mathbf{x}_n) = 1] \notin \Pr[D(\mathbf{u}_n) = 1] \pm \epsilon$. We say that D is an ϵ-distinguisher for a set $S \subseteq \{0,1\}^n$ if D is an ϵ-distinguisher for the uniform distribution over that set.*

We will repeatedly use reductions that map an input x for problem L to an input y of problem $\bar{L}$ where y is of the form xa, for some $a \in \{0,1\}^{|y|-|x|}$. We call these reductions input extending reductions:

Definition 3.2 (input-extending reductions)**.** *An* input-extending Karp reduction *of problem L to problem $\bar{L}$ is a function f such that for every $x \in \{0,1\}^*$:*

1. $x \in L \iff f(x) \in \bar{L}$.
2. $f(x)_{1,...,|x|} = x$.

All reductions in this paper will be by Karp reductions unless explicitly stated otherwise, and thus we will usually just refer to input-extending reductions.

When a reduction of f to $\bar{f}$ maps instances of length n_0 to instances of length $g(n_0) > n_0$, the function $\bar{f}$ is not necessarily defined on input lengths that are not of the form $g(n_0)$ in a non-trivial way. We will redefine $\bar{f}$ such that on any input x of length $n \in [g(n_0), g(n_0+1))$ it simply computes $f_{g(n_0)}$ on the $g(n_0)$-bit prefix of x.

Definition 3.3 (natural S-extension)**.** *Let $f\colon \{0,1\}^* \to \{0,1\}^*$ and $g\colon \mathbb{N} \to \mathbb{N}$ and denote $S = \{g(n_0) : n_0 \in \mathbb{N}\}$. We define a function $\bar{f}\colon \{0,1\}^* \to \{0,1\}^*$ called the* natural S-extension of f *as follows. On inputs of length that is in S, the function $\bar{f}$ agrees with f; and for every input length $n \notin S$ and every $x \in \{0,1\}^n$ it holds that $\bar{f}(x) = f(x_1, ..., x_m)$, where where m is the largest integer in S that is smaller than n.*

3.1 Complexity classes

We will sometimes abuse notation by denoting $\Pi \notin \text{i.o.}\mathcal{BPTIME}[T]$ for a function $\Pi\colon \{0,1\}^* \to \{0,1\}^*$ with multiple output bits. This should be interpreted as saying that for every probabilistic algorithm A running in time T and every sufficiently large $n \in \mathbb{N}$ there exists $x \in \{0,1\}^n$ such that $\Pr[A(x) = \Pi(x)] < 2/3$.

We recall standard definitions of average-case simulation of a problem $L \subseteq \{0,1\}^*$. The following definition refers to average-case simulation over an arbitrary distribution and over the uniform distribution.

Definition 3.4 (average-case simulation)**.** *Let $L \subseteq \{0,1\}^*$, let $\beta\colon \mathbb{N} \to [0,1)$, let $\mathcal{C}$ be a complexity class.*

1. *We say that $L \subseteq \mathsf{heur}_{1-\beta}\text{-}\mathcal{C}$ if for every polynomial-time samplable ensemble $\mathbf{x} = \{\mathbf{x}_n\}_{n\in\mathbb{N}}$ of distributions, where $\mathbf{x}_n$ is a distribution over n-bit inputs, there exists $L' \in \mathcal{C}$ such that for every sufficiently large $n \in \mathbb{N}$ it holds that $\Pr_{x\sim\mathbf{x}_n}[C(x) = L(x)] \geq 1 - \beta(n)$.*

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

3.1.1 Uniform circuits

Unless stated otherwise, the circuits that we consider in this work are Boolean circuits of fan-in two with gates computing the NAND function.

We define the depth of a gate g in a circuit to be the length of the shortest path from g to the set of input gates. Throughout the paper we will always assume that circuits are layered, meaning that all gates of depth i have incoming wires only from gates of depth $i-1$. The layer of input gates is defined to be of depth 0.

Definition 3.5 (logspace-uniform circuits)**.** *We say that a circuit family $\{C_n\colon \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ of size $T(n)$ is logspace-uniform if there exists a machine M that on input 1^n runs in space $O(\log(T(n)))$ and prints C_n. For two functions $T(n)$ and $d(n)$, we denote the class of languages computable by logspace-uniform circuits of size $T(n)$ and depth $d(n)$ by* lu-$\mathcal{CKT}[T,d]$.

Note that the foregoing definition is equivalent to a definition asserting that M recognizes the adjacency relation of C_n (i.e., receives the indices of three gates u, v, w and decides whether or not u and v feed into w). The definition of logspace-uniformity extends naturally to families of formulas, rather than circuits, as in Section 7.

3.2 Error-correcting codes

We first recall the well-known self-correcting properties of low-degree polynomials.

Theorem 3.6 (self-correction of low-degree polynomials (cf. [GS92; Sud95])**.** *Let $\delta < 1/3$, let $d, m \in \mathbb{N}$, and let $\mathbb{F}$ be a finite field such that $d < |\mathbb{F}|/3$. There exists an algorithm* RM-Dec *that, given $x \in \mathbb{F}^m$ and a description of $\mathbb{F}$ and oracle access to a function $f : \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial P of total degree d, runs in time $\mathrm{poly}(d, m, \log(|\mathbb{F}|))$ and outputs $P(x)$ with probability $2/3$. Furthermore:*

1. *If f itself has total degree d, then* RM-Dec *outputs $f(x) = P(x)$ with probability 1.*
2. *The algorithm* RM-Dec *makes $d + 1$ non-adaptive queries such that the marginal distribution of each query is uniform in $\mathbb{F}^m$.*

As usual, the error probability of RM-Dec can be decreased at an exponential rate by repeating the algorithm and taking the most common output.

We now recall the standard definition of approximately locally list-decodable codes, and state two well-known constructions.

Definition 3.7 (locally list decodable codes)**.** *We say that* $\mathsf{Enc} : \Sigma^N \to \Sigma^M$ *is* $(1-\delta)$-approximately locally list-decodable from agreement ρ in time t and with output-list size L *if there exists a randomized oracle machine* $\mathsf{Dec} : [N] \times [L] \to \Sigma$ *running in time t that satisfies the following. For every $z \in \Sigma^M$ such that $\Pr_{i \in [M]}[z_i = \mathsf{Enc}(x)_i] \geq \rho$ for some $x \in \Sigma^N$ there exists $a \in [L]$ and a set $Q \subseteq [N]$ of density $|Q|/N \geq 1 - \delta$ for which the following holds: For every $q \in Q$ we have that $\Pr[\mathsf{Dec}^z(q, a) = x_q] \geq 2/3$, where the probability is over the internal randomness of* Dec*. When $\delta = 0$ we simply say that* Enc *is* locally list-decodable from agreement ρ in time t and with output-list size L*.*

Theorem 3.8 (a locally list-decodable code [STV01]; see, e.g., [Vad12, Theorem 7.61])**.** *For every $\epsilon : \mathbb{N} \to (0,1)$ such that $\epsilon(N)$ is computable in space $O(\log(N))$, there exists a systematic*[20] *code* $\mathsf{Enc} : \{0,1\}^N \to \{0,1\}^{\bar{N}}$*, where $\bar{N} = \mathrm{poly}(N/\epsilon)$, such that* Enc *is computable in space $O(\log(N/\epsilon))$ and is locally list decodable from agreement $1/2 + \epsilon(N)$ in time $\mathrm{poly}(\log(N), 1/\epsilon)$ and with list size $\mathrm{poly}(\log(N)/\epsilon)$.*

The upper-bound on the space complexity of the code is not explicitly stated in [STV01; Vad12], but it follows immediately from the construction. (Recall that the code is a concatenation of the Reed-Muller code and of the Hadamard code; both codes are linear, and the RM code is defined over a field of size at most $\mathrm{poly}(N/\epsilon)$.) The fact that the code is systematic is also not stated there, and it follows by the fact that the Reed-Muller and the Hadamard code are both systematic, and by the definition of the Hadamard code.

We will use the following uniform direct product theorem, due to Impagliazzo *et al.* [IJK+10], who proved that the direct product code is approximately locally list decodable by uniform algorithms.

Definition 3.9 (direct product)**.** *For any function $f : \{0,1\}^* \to \{0,1\}^*$, we denote by $f^{\otimes k}$ the k-wise* direct product *$f^{\otimes k}(x_1, \ldots, x_k) = f(x_1), \ldots, f(x_k)$. The k-wise* direct product code *takes as input the truth-table of f, and outputs the truth-table of $f^{\otimes k}$.* [21]

Theorem 3.10 (the direct product code is approximately locally list-decodable; see [IJK+10, Theorem 1.6])**.** *There is a constant c such that for every $k \in \mathbb{N}$ and $\epsilon, \delta \in (0,1)$ such that $\epsilon > e^{-\delta k/c}$ the following holds. The k-wise direct product code is $(1-\delta)$-approximately locally*

[20] Recall that a code is *systematic* if the message always appears in the beginning of the codeword.

[21] The notion of "truth-table" here refers also to functions with multiple output bits, and just means the sequence of evaluations of the function on all inputs of a certain length.

list-decodable from agreement ϵ with list size $O(\epsilon)$ such that the decoder satisfies the following: Given a circuit C' that computes the codeword correctly on ϵ of the entries, the decoder is a uniform randomized $\mathcal{NC}^0$ algorithm (with one C'-oracle gate) that outputs with probability $\Omega(\epsilon)$ a circuit C that computes the original function on $1-\delta$ of the entries such that C is an $\mathcal{AC}^0$ circuit of size $\mathrm{poly}(n, k, \log(1/\delta), 1/\epsilon)$ *(with $O((\log(1/\delta)/\epsilon)$ of C'-oracle gates).*

3.3 Instance checkers and tolerant instance checkers

We first recall the definition of instance checkers, and then present two variations on this definition and prove some auxiliary results about these variations. In fact, we generalize even the first and standard definition, from the usual case of languages to the case of functions that might have multiple output bits, in a straightforward way.

Definition 3.11 (instance checker)**.** *A probabilistic oracle machine M is an* instance checker *for a function $f\colon \{0,1\}^* \to \{0,1\}^*$ if for every $x \in \{0,1\}^*$ the following holds:*

1. **Completeness:** $\Pr[M^f(x) = f(x)] = 1$.
2. **Soundness:** *For every $f'\colon \{0,1\}^* \to \{0,1\}^*$ we have that* $\Pr[M^{f'}(x) \in \{f(x), \perp\}] \geq 1/2$.

If on any input x the queries that M makes are of length $|x|$, we say that M is a same-length instance-checker.

As mentioned in Section 2, we also introduce refined notions of instance checkers that we will use to preserve strong notions of hardness – such as hardness on average and on all input lengths – under reductions. We now present the definitions of the refined instance checkers and prove a few simple auxiliary claims; we defer to Appendix A the proofs that these refined instance checkers help preserve strong notions of hardness under reductions.

3.3.1 Tolerant instance checkers

Informally, a tolerant same-length instance checker satisfies the same soundness condition as a standard same-length instance checker, but for completeness, we only require that when given an oracle that computes the language on $1-\delta$ of the inputs, the tolerant instance checker succeeds on computing the language on $1-\delta'$ of the inputs (on the δ' fraction of bad inputs it outputs "$\perp$", with high probability).

Definition 3.12 (tolerant instance checker)**.** *For $\delta, \delta' : \mathbb{N} \to [0,1]$, a (δ, δ')-*tolerant same-length instance checker *for $f\colon \{0,1\}^* \to \{0,1\}^*$ is a probabilistic oracle machine M such that*

1. **Tolerant completeness:** *For any $n \in \mathbb{N}$ and function $\tilde{f}\colon \{0,1\}^n \to \{0,1\}^*$ such that $\Pr_{x\in\{0,1\}^n}[\tilde{f}(x) = f(x)] \geq 1-\delta(n)$, the probability over $x \in \{0,1\}^n$ that $\Pr[M^{\tilde{f}}(x) = f(x)] \geq 2/3]$ is at least $1-\delta'(n)$.*
2. **Soundness:** *For any $n \in \mathbb{N}$ and $x \in \{0,1\}^n$ and $f'\colon \{0,1\}^n \to \{0,1\}^*$ we have that* $\Pr[M^{f'}(x) \in \{f(x), \perp\}] \geq 2/3$.

In this paper we will need the following property of tolerant instance checkers: If f has a tolerant instance checker, then its k-wise direct product $f^{\otimes k}$ has a tolerant instance checker that tolerates much higher levels of corruption.

Claim 3.12.1 (tolerant instance checkers for direct product problems)**.** *There exists a constant $c \geq 1$ such that for every $\delta, \delta', \epsilon : \mathbb{N} \to [0,1]$ the following holds. If $f\colon \{0,1\}^* \to \{0,1\}^*$ is (δ, δ')-tolerant instance checkable in time $T(n)$ and $k \geq c \cdot (\log(1/\epsilon)/\delta)$, then $f^{\otimes k}$ has a $(1-\epsilon, 1-\epsilon')$-tolerant instance checker that runs in time $T \cdot \mathrm{poly}(n, k, 1/\epsilon, \log(1/\delta), \log(1/\delta'))$, for $\epsilon' = (1-20\delta')^k$.* [22]

Proof. By Theorem 3.10, and since $k = \Omega(\log(1/\epsilon)/\delta)$, there is a probabilistic algorithm B that given as oracle a function C' that agrees with $f^{\otimes k}$ on at least ϵ of the inputs, outputs with probability $\Omega(\epsilon)$ a circuit C that computes L correctly on $1-\delta$ of the inputs. The algorithm B is a uniform randomized $\mathcal{NC}^0$ oracle circuit (with one oracle gate to C'), and the produced circuit C is an $\mathcal{AC}^0$ circuit of size $\mathrm{poly}(n, k, \log(1/\delta), 1/\epsilon)$ (with $O((\log(1/\delta)/\epsilon)$ oracle gates to C').

Let M be the (δ, δ')-tolerant instance checker for L. Using B and M we construct the tolerant instance checker for $L^{\otimes k}$. To do so, we first construct an algorithm M' that gets as input $x \in \{0,1\}^n$ and oracle access to $A\colon (\{0,1\}^n)^k \to \{0,1\}^r$ for some $r = r(n,k)$, and satisfies the following:

1. (Tolerant completeness:) If A agrees with $f^{\otimes k}$ on at least ϵ of the inputs, then for a $(1-20\delta')$ fraction of $x \in \{0,1\}^n$ it holds that $M'(x) = f(x)$ with probability at least $2/3$.

2. (Soundness:) For any A and any x, with probability at least $2/3$ it holds that $M'(x) \in \{f(x), \perp\}$.

To get the desired tolerant instance checker for $f^{\otimes k}$ we simply reduce the error probability of M' to $1/(10k)$ and run the procedure separately on each of the k inputs (see Remark 3.15). Indeed, the probability over $x_1, ..., x_k \in (\{0,1\}^n)^k$ that $\forall i, \Pr[M'(x_i) = f(x_i)] \geq 2/3$ is at least $(1-20\delta')^k = \epsilon'$.

We therefore proceed to describe the algorithm M'. Given an input $x \in \{0,1\}^n$ and an oracle $A : (\{0,1\}^n)^k \to \{0,1\}^r$, the algorithm M' operates as follows:

1. Run B with oracle access to A for $t = O(\log(1/\delta')/\epsilon)$ times, to obtain circuits $C_1, \ldots, C_t$.

2. For $j \in [t]$:

   (a) Run M on input x with soundness error $1/(10t)$ and using C_j^A as its oracle. If M outputs a value $b \neq \perp$ then output b and abort.

3. If the algorithm did not abort so far then output $\perp$ and abort.

Note that M' runs in time

$$\tilde{O}(t) \cdot \mathrm{poly}(n, k, \log(1/\delta), 1/\epsilon) \cdot T = T \cdot \mathrm{poly}(n, k, 1/\epsilon, \log(1/\delta), \log(1/\delta'))\ ,$$

and that reducing its error probability to $1/O(k)$ to obtain the final algorithm only increases the time complexity by $O(\log(k)) = O(\mathrm{loglog}(1/\epsilon) + \log(1/\delta))$.

Soundness. Fix an oracle A and an input x. By the soundness condition of M, for j, the probability that $M^{C_j^A}(x) \notin \{f(x), \perp\}$ is $1/(10t)$. By a union bound, with probability at least 0.9 the checker M' output either $f(x)$ or $\perp$.

[22] This statement slightly abuses notation, since even when referring to the instance checker for $f^{\otimes k}$, we still think of the functions ϵ and δ and δ' as functions of n (rather than of the actual input length $k \cdot n$).

Tolerant completeness. Fix an oracle $A : (\{0,1\}^n)^k \to \{0,1\}^r$ such that $\Pr_{x_1,\ldots,x_k}[A(x_1,\ldots,x_k) = f^{\otimes k}(x_1,\ldots,x_k)] \geq \epsilon$. Consider a random choice of $x \in \{0,1\}^n$ and (independently) of coins for B in the first step of M'.

By Theorem 3.10 and our setting of t, with probability at least $1-\delta'$ over random coins of B there exists at least one $j^* \in [t]$ such that $C^A_{j^*}$ agrees with f on at least $(1-\delta)$ of the inputs. In this case, by the tolerant completeness of M, for a $(1-\delta')$ fraction of $x \in \{0,1\}^n$ it holds that $\Pr[M^{C^A_{j^*}}(x) = f(x)] \geq 1 - 1/(10t)$. By a union bound, with probability at least $1-2\delta'$ over random coins for B and over $x \in \{0,1\}^n$ it holds that $\Pr[M^{C_{j^*}}(x) = f(x)] \geq 1 - 1/(10t)$.

Thus, by Markov's inequality, there exists a set $X \subseteq \{0,1\}^n$ of density at least $1-20\delta'$ such that for every $x \in X$, with probability 0.9 over the random coins for B it holds that $\Pr[M^{C_{j^*}}(x) = f(x)] \geq 1 - 1/(10t)$. Overall we get that for $x \in X$ the probability of the procedure succeeding is greater than $2/3$ as required. ■

3.3.2 Strongly tolerant instance checkers

We also consider a finer variant of tolerant same-length instance checkers that satisfies a stronger requirement: Namely, there is a equipartition of the domain such that the tolerant completeness condition holds within each set in the partition.

Definition 3.13 (strongly tolerant instance checker)**.** *For $\delta : \mathbb{N} \to [0,1]$ and $\ell(n) \leq n$, a ℓ-strongly δ-tolerant same-length instance checker for $f\colon \{0,1\}^* \to \{0,1\}^*$ is a probabilistic oracle machine M such that:*

1. **Strongly tolerant completeness:** *For any $x \in \{0,1\}^n$, which we parse as $x = (y,z) \in \{0,1\}^{\ell(n)} \times \{0,1\}^{n-\ell(n)}$, and any function $\tilde{f}\colon \{0,1\}^n \to \{0,1\}^*$ that satisfies $\Pr_{z' \in \{0,1\}^{n-\ell(n)}}[\tilde{f}(yz') = f(yz')] \geq 1-\delta(n)$, it holds that $\Pr[M^{\tilde{f}}(x) = f(x)] \geq 2/3$.*
2. **Soundness:** *For any $n \in \mathbb{N}$ and $x \in \{0,1\}^n$ and $f'\colon \{0,1\}^n \to \{0,1\}^*$ we have that $\Pr[M^{f'}(x) \in \{f(x), \perp\}] \geq 2/3$.*

A strongly tolerant instance checker is in particular a tolerant instance checker, using a simple averaging argument to claim that the strongly tolerant condition holds in many sets in the equipartition.

Fact 3.14 (strongly tolerant instance checker ⇒ instance checker)**.** *Let M be an ℓ-strongly μ-tolerant same length instance checker for f. Then, for any $\delta \in (0,1)$ it holds that M is a $(\delta, \delta/\mu)$-tolerant same length instance checker for f.*

Proof. Let $\tilde{f}$ such that $\Pr_{x \in \{0,1\}^n}[\tilde{f}(x) \neq f(x)] \leq \delta(n)$. Let

$$\mathsf{BAD} = \left\{ y \in \{0,1\}^{\ell(n)} : \Pr_{z \in \{0,1\}^{n-\ell(n)}}[\tilde{f}(yz) \neq f(yz)] \geq \mu \right\},$$

and note that $\Pr_y[y \in \mathsf{BAD}] \leq \delta/\mu$, by Markov's inequality. Then,

$$\begin{aligned}
&\Pr_x\left[\Pr[M^{\tilde{f}}(x) \neq f(x)] > 1/3\right] \\
&= \mathbb{E}_y\left[\Pr_z\left[\Pr[M^{\tilde{f}}(yz) \neq f(yz)] > 1/3\right]\right] \\
&\leq \Pr_y[y \in \mathsf{BAD}] + \Pr_{y,z}\left[\Pr[M^{\tilde{f}}(yz) \neq f(yz)] > 1/3 | y \notin \mathsf{BAD}\right],
\end{aligned}$$

which is at most δ/μ, relying on the fact that M is a ℓ-strongly μ-tolerant same-length instance checker (to deduce that the rightmost term is zero). ■

Remark 3.15. *As usual, the completeness and soundness errors in all definitions of instance checkers in this section can be reduced at an exponential rate. That is, to get error $2^{-\ell}$ we repeat the procedure for $\Theta(\ell)$ times and output $\sigma \in \{0,1\}$ if and only if a majority of the invocations returned σ, otherwise we output $\perp$.*

4 Technical tools

In this section we construct some of the technical tools that will be used throughout the paper. In Section 4.1 we construct two instance-checkable problems, one that is complete for linear space and one that is (essentially) complete for the class of logspace-uniform circuits. Then, in Section 4.2 we extend the targeted HSG from [CT21] by observing that it has two additional properties that are useful for us.

4.1 Instance checkable problems in $\mathcal{PSPACE}$ and beyond

We first construct an instance-checkable problem that is complete for linear space. To do so we recall a definition that abstracts the encoding of circuits as polynomials from [GKR15] (this definition appeared in [CT21]). Recall (from Section 3.1.1) that we refer to the layer of input gates as being of depth 0, to the layer of gates above it as being of depth 1, and so on.

Definition 4.1 (polynomial decomposition of a circuit; see [CT21, Definition 4.6])**.** *Let C be a circuit that has n input bits, fan-in two, size $T \geq n$, and depth d. For every $x \in \{0,1\}^n$, we call a collection of polynomials a* polynomial decomposition of $C(x)$ *if it meets the following specification:*

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

2. **(Circuit-structure polynomial.)** *For each $i \in [d]$, let $\Phi_i \colon H^{3m} \to \{0,1\}$ be the function such that $\Phi_i(\vec{w},\vec{u},\vec{v}) = 1$ if and only if the gate in layer i indexed by $\vec{w}$ is fed by the gates in layer $i-1$ indexed by $\vec{u}$ and $\vec{v}$. (If one of the elements $\vec{w},\vec{u},\vec{v}$ does not index a valid gate in the corresponding layer, then Φ_i outputs zero.) The polynomial $\hat{\Phi}_i \colon \mathbb{F}^{3m} \to \mathbb{F}$ can be any extension[23] of Φ_i.*

3. **(Input polynomial.)** *Let $\alpha_0 \colon H^m \to \{0,1\}$ represent the string $x0^{h^m-n} \in \{0,1\}^{h^m}$, where we identify H^m with $[h]^m$ (e.g., by lexicographic order). Let $\hat{\alpha}_0 \colon \mathbb{F}^m \to \mathbb{F}$ be the unique extension of individual degree $h-1$ of α_0 defined by*

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

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

[23] An *extension* of a function $f : H^m \to \mathbb{F}$ is any function $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ that agrees with f on $H^m \subseteq \mathbb{F}^m$.

4. **(Layer polynomials.)** *For each $i \in [d]$, let $\alpha_i \colon H^m \to \{0,1\}$ represent the values of the gates at the i^{th} layer[24] of C in the computation of $C(x)$ (with zeroes in locations that do not index valid gates), and let $\hat{\alpha}_i \colon \mathbb{F}^m \to \mathbb{F}$ be defined by*

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

5. **(Sumcheck polynomials.)** *For each $i \in [d]$ and for $j \in [2m]$, we define a polynomial $\hat{\alpha}_{i,j} \colon \mathbb{F}^{m+j} \to \mathbb{F}$ as follows:*

$$\hat{\alpha}_{i,0}(\vec{w}) = \hat{\alpha}_i(\vec{w}) = \sum_{\sigma_1,\ldots,\sigma_{2m} \in H} \hat{\Phi}_i(\vec{w},\sigma_{1,\ldots,m},\sigma_{m+1,\ldots,2m}) \cdot \left(1 - \hat{\alpha}_{i-1}(\sigma_{1,\ldots,m}) \cdot \hat{\alpha}_{i-1}(\sigma_{m+1,\ldots,2m})\right)$$

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

where $\sigma_{k\ldots,k+r} = \sigma_k, \sigma_{k+1}, \ldots, \sigma_{k+r}$. [25]

We stress that for any arithmetic setting $\mathbb{F}_p$ and $H \subseteq \mathbb{F}$ and m', m that meets the specification above, any collection $\left\{ \{\hat{\Phi}_i\}_{i\in[d]}, \{\hat{\alpha}_i\}_{i=0}^{d}, \{\hat{\alpha}_{i,j}\}_{i\in[d],j\in[2m]} \right\}$ that meets the specification above is a polynomial decomposition of $C(x)$.

Proposition 4.2 (polynomial decomposition of logspace-uniform circuits using universal circuits; see [CT21, Proposition 4.7])**.** *There exist two universal constants $c, c' \in \mathbb{N}$ such that the following holds. Let $\{C_n\}_{n\in\mathbb{N}}$ be a logspace-uniform family of circuits of size $T(n)$ and depth $d(n)$, and let $\gamma \colon \mathbb{N} \to (0,1)$ be a logspace-computable function such that $T(n)^{\gamma(n)} \geq \log(T(n))$. Then, there exists a logspace-uniform family of circuits $\{C'_n\}_{n\in\mathbb{N}}$ of size $T'(n) = O(T(n)^c)$ and depth $d'(n) = O(d(n) \cdot \log(T(n)))$ that computes the same Boolean function as $\{C_n\}$ such that for every $x \in \{0,1\}^n$ there exists a polynomial decomposition of $C'_n(x)$ satisfying:*

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

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

3. **(Base layer.)** *There is a logspace-uniform oracle circuit of size $\max\{n,h\} \cdot h^{c'}$ and depth $O(\log^2(T))$ that, when given oracle access to $x \in \{0,1\}^n$, computes the function $\hat{\alpha}_0$.*

4. **(Downward self-reducibility.)** *There is a logspace-uniform oracle circuit of size $h^{c'}$ and depth $O(\log^2(T))$ that computes $\hat{\alpha}_{i,2m}$ while querying $\hat{\alpha}_{i-1}$ twice on inputs in H^m. There is another logspace-uniform oracle circuit of size $h^{c'}$ and depth $O(\log T)$ that for $j \in [2m]$ computes $\hat{\alpha}_{i,j-1}$ while making h queries to $\hat{\alpha}_{i,j}$.* [26] *Lastly, $\hat{\alpha}_{i,0} \equiv \hat{\alpha}_i$.*

[24] That is, the function α_i represents the string that consists of the values of the gates at the i^{th} layer (padded with zeroes to be of length h^m), where we identify H^m with $[h]^m$ (say, by lexicographic order).

[25] Note that $\alpha_{i,2m}(\vec{w},\sigma_1,\ldots,\sigma_{2m}) = \hat{\Phi}_i(\vec{w},\sigma_{1,\ldots,m},\sigma_{m+1,\ldots,2m}) \cdot (1 - \hat{\alpha}_{i-1}(\sigma_{1,\ldots,m}) \cdot \alpha_{i-1}(\sigma_{m+1,\ldots,2m}))$, without any summation.

[26] More formally, this logspace-uniform oracle circuit gets as input $\vec{w} \in \mathbb{F}^m$ and $(\sigma_1,\ldots,\sigma_{2m}) \in \mathbb{F}^{2m}$ and $j \in [2m]$, and gets oracle access to $\hat{\alpha}_{i,j}$, and it outputs $\hat{\alpha}_{i,j-1}(\vec{w},\sigma_1,\ldots,\sigma_{j-1})$.

5. **(Sample-aided worst-case to rare-case reducibility.)** [27] *For each $i \in [d'(n)]$ and $j \in [2m]$, the total degree of $\hat{\alpha}_{i,j}$ is at most $\Delta = h \cdot \mathrm{polylog}(T)$. In particular, the Boolean function representing $\hat{\alpha}_{i,j}$ is sample-aided worst-case to ρ-rare-case reducible with error 2^{-h} by logspace-uniform circuits of size $h^{c'}$ and depth $\mathrm{polylog}(T)$, where $\rho = h^{-c} \cdot \mathrm{polylog}(T)$. (The same claims holds for $\hat{\alpha}_0$.)*

Furthermore, if the mapping of $(x,i) \in \{0,1\}^n \times [T(n)]$ to the value of the i^{th} gate in $C_n(x)$ is computable in space $O(\log(T))$, then we can evaluate each of the polynomials $\hat{\alpha}_{i,j}$ (for $i \in [d']$ and $j \in [2m]$), as well as $\hat{\alpha}_0$, in space $O(\log(T))$.

Proof. The only part of Proposition 4.2 that is not proved in [CT21] is the "furthermore" part, and thus we will prove it now. Recall that C_n is a logspace-uniform circuit, and let A be the $O(\log(T))$-space machine that prints C_n. Let $M = M_A$ be the $\mathrm{poly}(T) \times \mathrm{poly}(T)$ transition matrix of A.

The circuit C'_n first computes M, then raises it to the power $\mathrm{poly}(T)$ by repeated squaring; this yields the description of C_n. Then the circuit C'_n computes the Eval function on that description and with the input x. In more detail, the Eval function is computed in d iterative steps; in each step the input is the evaluations of the previous layer of $C_n(x)$ (starting from the bottom layer, which is the input x, and working up to the top layer) and the description of C_n, and each gate g in the current layer is computed as

$$g(x) = \bigvee_{u,v \in [T]} \left(\mathbf{1}_{(u,v)\sim g} \wedge \mathsf{NAND}(u(x), v(x)) \right) , \tag{4.1}$$

where u, v are (indices of) gates in C_n, and $\mathbf{1}_{(u,v)\sim g}$ is the bit in the description of C_n indicating whether or not u and v feed into g.

Computing each basic summand in logspace. Our first goal is to prove that we can compute the "basic summand" $\Phi_i(\vec{w}) \cdot (1 - \hat{\alpha}_{i-1}(\vec{u}) \cdot \hat{\alpha}_{i-1}(\vec{v}))$ that appears in the definition of the $\hat{\alpha}_{i,j}$'s in space $O(\log(T))$. That is,

Claim 4.2.1. *Given x, i and any $\vec{w} \in \mathbb{F}^m$ and $\vec{u}, \vec{v} \in H^m$, we can compute the value*

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

in space $O(\log(T))$.

Proof. The proof proceeds in three steps. We first argue that we can compute the value of any gate in $C'_n(x)$ (rather than only $C_n(x)$) in space $O(\log(T))$; specifically:

Claim 4.2.1.1. *The mapping of $(x,i) \in \{0,1\}^n \times \{0,1\}^{T'(n)}$ to the value of the i^{th} gate in $C'_n(x)$ is also computable in space $O(\log(T))$.*

Proof. The $O(\log(T))$-space machine gets input (x,i) and behaves as follows:

1. If i is an index of a gate that is an entry in M, it outputs the relevant entry, by applying the transition function of A to the two relevant states. Note that this can indeed be computed in space $O(\log(T))$.

[27] The notion of sample-aided worst-case to rare-case reducibility was introduced by Goldreich and Rothblum [GR17]. The terminology in the current text uses the formulation from [CT21, Definition 3.8].

2. If i is an index of a gate that is an entry in M^t for some power $t \leq O(\log(T))$, then this gate is associated with two potential configurations $\gamma, \gamma' \in \{0,1\}^{O(\log(T))}$ (i.e., the corresponding row and the column), and its value is 1 iff A reaches the configuration γ' when starting from configuration γ and running for 2^t steps. We simulate A for 2^t steps in space $O(\log(T))$ and output the corresponding value.

3. If i is an index of a gate g in the part that computes the Eval function on (C_n, x), by Eq. (4.1) we can compute it in space $O(\log(T))$ if we get oracle access to the evaluations of all gates u, v in the previous layer on x. By our hypothesis, the latter evaluations are computable in space $O(\log(T))$.

4. The last case is that i is an index of a gate performing an intermediary computation: This intermediary computation can be part of computing an entry in a multiplication of two matrices, or part of computing the value of a gate g when simulating the Eval function, as in Eq. (4.1). In both cases, the computation is the OR of at most $\text{poly}(T)$ gates, and the intermediary computation performed by gate i is an OR of a subset of these gates.[28] We can enumerate the gates in this subset from the index i and compute their disjunction in space $O(\log(T))$.

□

Next, we argue that the low-degree extension $\hat{\Phi}$ of the circuit-structure function Φ can also be computed in space $O(\log(T))$, as follows:

Claim 4.2.2. *The mapping of* $(x, i, \vec{w}, \vec{u}, \vec{v}) \in \{0,1\}^n \times [d'] \times (\mathbb{F}^m)^3$ *to* $\hat{\Phi}_i(\vec{w}, \vec{u}, \vec{v})$ *can be computed in space* $O(\log(T(n)))$.

Proof. Recall that the purpose of constructing C'_n from C_n as above was to have a circuit whose circuit-structure function Φ is very simple. While in [Gol18], where the construction was first presented, it is only asserted that Φ is a formula of size $\text{polylog}(T)$ that can be constructed in time $\text{polylog}(T)$, the actual construction in [Gol18, Section 3.4.2] is significantly more efficient than that, and in particular can be decided by an $\mathcal{NC}^1$ circuit constructed in time $\text{polylog}(T)$ and space $O(\log(T))$.

To see this, note that Φ on gates in the first part of C'_n is a trivial function, since this part has no wiring and just computes a fixed string (i.e., the transition matrix M of A). An explicit $\mathcal{AC}^0$ formula for evaluating Φ on gates in the second part (i.e., in the part performing repeated squaring) was presented in [Gol18, Section 3.4.2]. And as for the third part, to establish whether gates (u', v') feed into a gate w', we just need to determine whether the indices of u' and v' are the children of the index w' in the binary tree that computes the OR tree in Eq. (4.1).

Now, identically to the proof of [CT21, Claim 4.7.2], the arithmetization $\hat{\Phi}_i \colon \mathbb{F}^{3m} \to \mathbb{F}$ is computed by first computing the projection polynomials $\hat{\pi}_j \colon \mathbb{F} \to \{0,1\}$ on each of the $3m$ field elements, and then evaluating Φ as an arithmetic formula $\mathbb{F}_2^{3m \cdot \log(|H|)} \to \mathbb{F}_2$ on the resulting sequence of bits. Since each of these two steps can be done in space $O(\log(T))$, their composition can also be done in such space complexity.[29] □

[28] Recall that the circuit computes the OR function by a binary tree. Also, recall that in the matrices that we are multiplying, each row and each column have Hamming weight one.

[29] Each projection polynomial $\hat{\pi}_j \colon \mathbb{F} \to \{0,1\}$ is defined as the $(|H|-1)$-degree extension of the function π_j that maps each element in $H \subseteq \mathbb{F}$ to the j^{th} bit of its binary representation. Indeed, π_j can be computed in space $O(\log(T))$, and thus (using Lagrange's interpolation formula) so does $\hat{\pi}_j$.

Based on the two foregoing claims, we now show how to compute the summand in space $O(\log(T))$. Observe that we are evaluating $\hat{\alpha}_{i-1}$ at inputs in H^m, and thus these inputs correspond to indices of gates. In particular, $\hat{\alpha}_{i-1}(\vec{u})$ and $\hat{\alpha}_{i-1}(\vec{v})$ are in $\{0,1\}$ and represent the values of gates in $C'_n(x)$. By Claim 4.2.2, we can compute the values of these gates in space $O(\log(T))$, and by Claim 4.2.1.1, we can also compute $\hat{\Phi}_i$ in space $O(\log(T(n)))$. □

Using downward self-reductions. Observe that for every $i \in [d']$, computing the polynomial $\hat{\alpha}_i = \hat{\alpha}_{i,0}$ requires summing over $H^{O(m)} \leq \text{poly}(T)$ "basic summands" of the form $\hat{\Phi}_i(\vec{w},\vec{u},\vec{v}) \cdot (1 - \hat{\alpha}_{i-1}(\vec{u}) \cdot \hat{\alpha}_{i-1}(\vec{v}))$, where $\vec{u}$ and $\vec{v}$ are in H^m. Thus, relying on Claim 4.2.1, we can compute any such polynomial in space $O(\log(T))$. Also, $\hat{\alpha}_0$ is computable in space $O(\log(T))$, immediately by its definition in Definition 4.1.

Now, for $i \in [d']$ and $j \in [2m]$, the value of the polynomial $\hat{\alpha}_{i,j}$ at point $(\vec{w},\sigma_1,...,\sigma_j)$ is the summation of $H^{O(m)} \leq \text{poly}(T)$ summands of the form

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

By Claim 4.2.2 we can compute $\hat{\Phi}_i$ in space $O(\log(T))$, and by the discussion above we can compute $\hat{\alpha}_{i-1}$ in space $O(\log(T))$. Thus, any summand can be computed in such space, and so does $\hat{\alpha}_{i,j}$. ■

Relying on Proposition 4.2, we now present the construction of the same-length instance-checkable language.

Proposition 4.3 (a linear-space complete instance-checkable problem)**.** *There exists a problem $L \subseteq \{0,1\}^*$ that is complete for $\mathcal{SPACE}[O(n)]$ under linear-time reductions, and is same-length instance checkable in polynomial time.*

Proof. Let $L^{(0)} \subseteq \{0,1\}^*$ be a language that is complete for linear space under linear-time reductions.[30] Let $\{C_n : \{0,1\}^n \to \{0,1\}\}$ be a logspace-uniform family of circuits that decides $L^{(0)}$, where C_n is of size $T(n) = 2^{\Theta(n)}$ and depth $d(n) = O(n^2)$; recall that such a circuit family is obtained by the standard construction of repeatedly squaring the transition matrix of the linear-time machine that computes $L^{(0)}$. For a sufficiently small $\gamma(n) > 0$ that will be specified later, let $\{C'_n\}$ be the circuit family from Proposition 4.2 corresponding to $\{C_n\}$, where C'_n is of size $T'(n) = \text{poly}(T(n))$ and depth $d' = O(n^3)$, and consider the polynomial decompositions of $C'_n(x)$ (for any fixed $x \in \{0,1\}^n$).

For convenience, we think of all the polynomials $\hat{\alpha}_{i,j}$ in the decomposition as having the same domain $\mathbb{F}^{3m}$. (Originally, the domain of $\hat{\alpha}_{i,j}$ was $\mathbb{F}^{m+j}$, but we can define $\hat{\alpha}_{i,j}$ so that it ignores the last $2m - j$ elements in its input, without affecting any of the properties asserted in Proposition 4.2.)

The definition of L and its $\mathcal{SPACE}[O(n)]$-completeness. We define $L \subseteq \{0,1\}^*$ by the following procedure:

1. The inputs to L are of the form $(x,(i,j),\vec{w},k)$ where $x \in \{0,1\}^n$ and $(i,j) \in \{0,...,d'\} \times \{0,...,2m\}$ and $\vec{w} \in \mathbb{F}^{3m}$ and $k \in [\lceil \log(|\mathbb{F}|) \rceil]$.
2. The output of L is the k^{th} bit of the binary representation of $\hat{\alpha}_{i,j}(\vec{w})$. (When $i = 0$, the output is the k^{th} bit of $\hat{\alpha}_0(\vec{w})$, disregarding j.)

[30] For example, the language $\{(1^n,x,M) : M \text{ accepts } x \text{ using space } n\}$.

Note that for each input length $n \in \mathbb{N}$ to $L^{(0)}$, we defined L on a corresponding input length

$$n + \lceil \log(d'+1) \rceil + \lceil \log(2m+1) \rceil + 3m \cdot \lceil \log(|\mathbb{F}|) \rceil + \lceil \log\log(|\mathbb{F}|) \rceil = O(n) \ . \quad (4.2)$$

Also note that $L^{(0)}$ is reducible in linear time to L, since the polynomial $\hat{\alpha}_{d',0}$ is an extension of the function whose truth-table is the string $L^{(0)}(x) \circ 0^{T'-1} \in \{0,1\}^{T'}$. (To be specific, the reduction maps an input $x \in \{0,1\}^n$ to $L^{(0)}$ into an input $(x,(d',0),\vec{0},0)$ for L.)

Finally, the claim that L is computable in space $O(n)$ follows from the "furthermore" part of Proposition 4.2. Specifically, to invoke the "furthermore" part we just need to show that we can compute the mapping of (x,i) to the value of the i^{th} gate of $C_n(x)$ in space $O(\log(T))$. This holds because of the specific construction of C_n as a "repeated squaring" matrix; specifically, each gate g in C_n is the result of repeatedly squaring the transition matrix of the machine $M^{(0)}$ for $L^{(0)}$, applied to input x, for $t \leq O(\log(T))$ times. The gate g is associated with two potential configuration γ, γ' of the machine $M^{(0)}$, and evaluates to 1 if and only if running $M^{(0)}$ for 2^t steps starting from configuration γ yields the configuration γ'. Thus, to decide the value of $g(x)$ we can simply run $M^{(0)}$ for 2^t steps with configuration γ. [31]

Same-length instance-checkability. Let us first describe the instance checker A and then analyze it. Recall that the algorithm gets input $(x,(i,j),\vec{w},k)$. The instance checker will only make queries with x as the first input, and for every fixed (i',j') and $\vec{w}'$ it will iterate over all values of k to obtain the corresponding field element. Thus, without loss of generality, we think of the oracle as receiving queries of the form $(i',j',\vec{w}')$ and answering with a field element; the instance checker expects this field element to be $\hat{\alpha}_{i',j'}(\vec{w}')$.

We define the following randomized procedure, which we call query verification. Given input $(i',j',\vec{w}')$ where either $i' \geq 2$ or $j' < 2m$, we want to reduce the verification of the oracle's answer on this query to verifying the oracle's answer on another query. We invoke the downward self-reducibility algorithm for $\hat{\alpha}_{i',j'}$, which yields a set of at most h queries either to $\hat{\alpha}_{i',j'+1}$ or to $\hat{\alpha}_{i'-1}$. Without loss of generality, we assume that the number of queries is exactly h, denote them by $q_1,...,q_h \in \mathbb{F}^m$, and denote the function to which the queries are made by $\hat{\alpha}_{i'',j''}$.

Recall that the total degree of every $\hat{\alpha}_{i,j}$ is at most $\Delta = h \cdot \text{polylog}(T)$ (for every i,j). Consider the curve $p\colon \mathbb{F} \to \mathbb{F}^m$ of degree $h-1$ such that $p(u) = q_u$ for all $u \in [h]$, and for $u \in \{h+1,...,\Delta \cdot h\}$ let $q_u = p(u)$; this yields a set $\{q_u = p(u) : u \in [\Delta \cdot h]\}$ of points on the curve p, where the first h points correspond to the queries of the downward self-reducibility algorithm. We query the oracle on $\{(i'',j'',q_u) : u \in [\Delta \cdot h]\}$, yielding answers $a_1,...,a_{\Delta \cdot h} \in \mathbb{F}$, and verify that the output of the downward self-reducibility algorithm (when given answers $a_1,...,a_h$) is consistent with the oracle's answer on query $(i',j',\vec{w}')$ (in case of inconsistency, we output $\perp$). Now, let q be the unique polynomial of degree $h \cdot \Delta - 1$ such that $q(p(u)) = a_u$ for all $u \in [h \cdot \Delta]$ (see Eq. (4.3) below); we choose a random $\sigma \in \mathbb{F}$, and verify that the value $q(\sigma)$ agrees with

[31] To be more precise, some gates in C_n are associated with potential configurations, whereas other gates are intermediary values in computing the squaring of a matrix M' (where M' is the initial transition matrix raised to some power $t' \leq t$). The value of the latter type of gates can be computed in space $O(\log(T))$, relying on the fact that we can compute each entry in M' itself in $O(\log(T))$, and that values of intermediary gates when squaring M' can be computed in space $O(\log(T))$.

the oracle on query $(i'', j'', p(\sigma))$. Specifically, we compute q as follows

$$q(\sigma) = \sum_{u \in [h \cdot \Delta]} a_u \cdot \delta^{(h \cdot \Delta)}_{p(u)}(\sigma) \ , \quad \left(\text{where } \delta^{(h \cdot \Delta)}_{p(u)}(\sigma) = \prod_{\sigma' \in [h \cdot \Delta] \setminus \{p(u)\}} \frac{\sigma - \sigma'}{p(u) - \sigma'} \right) \tag{4.3}$$

and the target of the reduction is $(i'', j'', p(\sigma))$.

The instance checker recursively calls the query verification procedure, starting from input $(i, j, \vec{w})$ and ending in the base case when $i' = 1$ and $j' = 2m$. In the base case the instance checker runs the downward self-reducibility algorithm, answers its queries to $\hat{\alpha}_0$ by computing $\hat{\alpha}_0$ by itself, and verifies that the answer of the downward self-reducibility algorithm matches the oracle's answer on $(1, 2m, \vec{w})$. If all verifications passed, the instance checker outputs the oracle's answer on the input $(i, j, \vec{w})$.

Time complexity of the instance-checker. Note that there are at most $2m \cdot d'$ recursive steps, and that each step can be computed in time at most

$$\log(|\mathbb{F}|)^{O(1)} \cdot \left(\max\{n, h\} \cdot h^{2c'} + h^{O(c')} + (h \cdot \Delta)^2 \right) \leq (n \cdot h)^{O(c')} \ ,$$

where the $h^{O(c')}$ term accounts for the time for computing the curve p, and the O-notations hide universal constants. Thus, the running time of the instance checker is at most

$$m \cdot d' \cdot (n \cdot h)^{c''}$$

for some universal constant $c'' > 1$.

Completeness and soundness. We first argue that if all answers given by the oracle are correct (i.e., the oracle answers by deciding L on the given query), then the instance checker outputs the correct value with probability 1. This amounts to showing that all the verifications of the instance checker pass. Since the answers of the downward self-reducibility algorithm will always match the oracle's answers, the only thing that we need to show is that Eq. (4.3) matches the oracle's answer, which is $\hat{\alpha}_{i'', j''}(p(\sigma))$. To see this, note that $a_u = \hat{\alpha}_{i'', j''}(p(u))$, and plugging this into Eq. (4.3) we get

$$q(\sigma) = \sum_{u \in [h]} \hat{\alpha}_{i'', j''}(p(u)) \cdot \delta^{(h \cdot \Delta)}_{p(u)}(\sigma) \ .$$

Note that q is a polynomial $\mathbb{F} \to \mathbb{F}$ of degree $h \cdot \Delta - 1$, that the polynomial $\hat{\alpha}_{i'', j''} \circ p \colon \mathbb{F} \to \mathbb{F}$ is of degree $(h-1) \cdot \Delta < h \cdot \Delta - 1$, and that both polynomials agree on the inputs $[h \cdot \Delta] \subset \mathbb{F}$. It follows that $q \equiv \hat{\alpha}_{i'', j''} \circ p$, and hence on input σ its output equals the value $\hat{\alpha}_{i'', j''}(p(\sigma))$.

We now turn to the case where the oracle answered at least one query incorrectly (i.e., gave an answer that is different from L on the given query). Our goal is to show that in this case, with high probability the instance checker either outputs $\perp$ or outputs the correct answer. We say that an input $(i', j', \vec{w'})$ is good if the oracle's answer on $(i', j', \vec{w'})$ is $\hat{\alpha}_{i', j'}(\vec{w'})$. Without loss of generality, we assume the following:

1. The input $(i, j, \vec{w})$ to the instance checker is not good. (Otherwise, the instance checker may only output the right answer or $\perp$.)

2. In each query verification step starting with input $(i', j', \vec{w'})$, the answers $a_1, ..., a_h$ cause the downwards self-reducibility algorithm to output the oracle's answer on $(i', j', \vec{w'})$. (Otherwise, the instance checker outputs $\perp$.)

3. In each query verification step, the oracle's answer on query $(i'', j'', p(\sigma))$ equals $q(\sigma)$. (Otherwise the instance checker outputs $\perp$.)

The main claim in the analysis is the following:

Claim 4.3.1. *For any query verification step starting with an input $(i', j', \vec{w}')$ that is not good, with probability at least $1 - T^{-(c/2)\cdot\gamma}$ the target $(i'', j'', p(\sigma))$ of the reduction is also not good.*

Proof. We first claim that there exists $u \in [h]$ such that $(i'', j'', p(u))$ is not good. To see this, note that if $a_1, ..., a_h$ are good, then the downward self-reducibility outputs $\hat{\alpha}_{i',j'}(\vec{w}')$, which is different than the oracle's answer on $(i', j', \vec{w}')$ (because the latter input is not good).

Consequently, the polynomial q in Eq. (4.3) disagrees with the polynomial $\hat{\alpha}_{i'',j''} \circ p$ (on u). Since both polynomials are of degree $\Delta \cdot h - 1 = \tilde{O}(T^{\gamma/3})$, and the field is of size at least $T^{\gamma \cdot c}$, with probability at least $1 - T^{-(c/2)\cdot\gamma}$ over choice of σ we have that $q(\sigma) \neq \hat{\alpha}_{i'',j''} \circ p(\sigma)$. Now, recall that we assumed (wlog) that $q(\sigma)$ equals the oracle's answer on $(i'', j'', p(\sigma))$. Thus, whenever the choice of σ satisfies the event above, it follows that $(i'', j'', p(\sigma))$ is not good. □

By a union-bound, with probability at least $1 - 2m \cdot d' \cdot T^{-(c/2)\cdot\gamma}$ it holds that the query $(1, 2m, \vec{w})$ in the base case is not good. Whenever this happens, the answer of the downward self-reducibility in this case (which equals $\hat{\alpha}_{1,2m}(\vec{w})$) does not match the oracle's answer on $(1, 2m, \vec{w})$, and the instance checker outputs $\perp$.

Specifying the parameters. We have yet to specify the value of $\gamma(n)$. The argument above establishes the following bounds (we use the facts that $h = O(T^{\gamma/6})$ and that $m = O(1/\gamma)$, and for simplicity we hide a multiplicative constant in each bound):

$$\begin{aligned} \text{Running time of the instance checker:} &\quad (d'/\gamma) \cdot (n \cdot T^{\gamma/6})^{c''} \\ \text{Probability that the instance checker errs:} &\quad (d'/\gamma) \cdot T^{-(c/2)\cdot\gamma} \end{aligned}$$

Recall that $T = T(n) = 2^{\Theta(n)}$. We choose $\gamma = \gamma(n) = O(\log(n)/n)$ such that $T^{(c/2)\cdot\gamma} > n^6$. This makes the error bound at most $1/n$, bounds the running time by a universal polynomial in n, and satisfies the hypotheses of Proposition 4.2 that $T^{\gamma} \geq \log(T)$ and that γ is a logspace-computable function. ■

The following result extends Proposition 4.3 by asserting that there is an instance-checkable problem that is complete for the class of logspace-uniform circuits of bounded depth, and proving several new properties of the problem that will be useful for us.

Proposition 4.4 (an instance-checkable problem that is complete for logspace-uniform circuits)**.** *For $T, d \colon \mathbb{N} \to \mathbb{N}$ such that $n \leq T(n) \leq 2^{O(n)}$, let $L^{(0)} \in \mathsf{lu}\text{-}\mathcal{CKT}[T, d]$. Then, $L^{(0)}$ is reducible in linear time to a problem $L \in \mathsf{lu}\text{-}\mathcal{CKT}[\mathrm{poly}(T), O(d \cdot \log(T)^4)]$ such that L is same-length instance checkable in time $\mathrm{poly}(d, \log(T), n)$. The reduction is input-extending and maps inputs of length n to inputs of length $n + c \cdot \log(T)$, where $c > 1$ is a universal constant.*

Furthermore, for a constant $c' \leq 1 + c_0 \cdot \log(T)/n$, where $c_0 > 1$ is universal, there is a function $F \colon \{0,1\}^* \to \{0,1\}^*$ such that:

1. *F is computable by logspace-uniform circuits of size $\mathrm{poly}(T)$ and depth $O(d \cdot \log(T)^4)$.*
2. *There is an input-extending reduction of deciding $L^{(0)}$ to computing F that maps inputs of length n to inputs of length $c' \cdot n$.*

3. *F has an (n/c')-strongly $(1/10)$-tolerant same-length instance checker running in time* $\mathrm{poly}(d, \log(T), n)$.

Proof. We follow the proof of Proposition 4.3, while explaining the necessary changes, and we defer showing the "furthermore" part to the end of the current proof.

In the original proof we started from a problem $L^{(0)}$ computable in linear space, constructed a logspace-uniform circuit family $\{C_n\}$ of size $2^{O(n)}$ and depth $O(n^2)$ that decides $L^{(0)}$ and that has a specific useful structure (i.e., it comes from applying repeated squaring to the linear-space machine that decides $L^{(0)}$), and carried out the proof from the starting point of having the circuit family $\{C_n\}$.

In the current proof, we start from a problem $L^{(0)}$ having a circuit family $\{C_n\}$ with more general size and depth bounds, and in addition $\{C_n\}$ does not necessarily have the specific useful structure (of repeated squaring). However, we are not trying to prove that the related instance-checkable language L is decidable in linear space, but rather only that it is decidable by logspace-uniform circuits of size $\mathrm{poly}(T)$ and depth $d \cdot \mathrm{polylog}(T)$.

We will use Proposition 4.2 with the circuit family that decides $L^{(0)}$ and with the parameter value $\gamma(n) = O(\log(d' \cdot n)/\log(T))$ (recall that $d'(n) = O(d(n) \cdot \log(T(n)))$); this parameter choice guarantees that the probability of error is at most $1/n$, and that the running time is at most $\mathrm{poly}(d', n)$. Note that for this parameter value we have that $m = O(1/\gamma) = O(\log(T)/\log(d' \cdot n)) < O(\log(T))$.

Recall that the language L, resulting from the application of Proposition 4.2, has inputs of the form $(x, (i,j), \vec{w}, k)$ where $x \in \{0,1\}^n$, $(i,j) \in \{0, ..., d'\} \times \{0, ..., 2m\}$, and $\vec{w} \in \mathbb{F}^{3m}$ and $k \in [\log(|\mathbb{F}|)]$. We have $(x, (i,j), \vec{w}, k) \in L$ if and only if the k^{th} bit of the binary representation of $\hat{\alpha}_{i,j}(\vec{w})$ is 1.

First observe that Eq. (4.2) still holds, due to our hypothesis that $T(n) \leq 2^{O(n)}$ (and using the trivial bound $d(n) \leq T(n)$ and the fact that $|\mathbb{F}|^m = \mathrm{poly}(T)$). Thus, $L^{(0)}$ is reducible to L in linear time, where n-bit inputs are mapped to inputs of length $n + O(m \cdot \log(|\mathbb{F}|)) = n + O(\log(T))$ and the O-notation hides a universal constant; this implies the claimed bound on c'.

To upper-bound the complexity of L, note that by Definition 4.1, we can compute $\hat{\alpha}_0$ by a circuit of size $\mathrm{poly}(T)$ and depth $O(\log(T))$; and by Proposition 4.2, we can compute each $\hat{\alpha}_{i,j}$ on all values in $\mathbb{F}^m$ given access to $\hat{\alpha}_{i,j-1}$ or to $\hat{\alpha}_{i-1}$ by a circuit of size $\mathrm{poly}(T)$ and depth $O(\log(T)^2)$. Thus, we can compute each $\hat{\alpha}_{i,j}$ iteratively in at most $2m \cdot d'$ steps, each step computable by a circuit of size $\mathrm{poly}(T)$ and depth $O(\log(T)^2)$. This yields the sought upper-bound on the complexity of L.

Finally, the proof of correctness for the instance-checker of L is identical to corresponding part in the proof of Proposition 4.3.

The "furthermore" part. We construct a function F based on L that satisfies the same properties asserted for L but also has a strongly tolerant instance checker. Loosely speaking, the function is defined by "bundling" the polynomials $\hat{\alpha}_{i,j}$ into one polynomial, for every fixed x, and outputting a field element rather than a single bit in the representation of a field element.

In more detail, we first assume that the prime size p of the field used above is larger than $(d \cdot \log(T) \cdot n)^{\bar{c}}$ for a sufficiently large universal constant $\bar{c} > 1$ that will be specified next. (Recall that $p \geq T^{c \cdot \gamma}$ and that we choose $\gamma = \Theta(\log(d \cdot \log(T) \cdot n)/\log(T))$ and can take the constant hidden inside the Θ-notation to be large enough.) For any

$x \in \{0,1\}^n$, let $P_x : \mathbb{F}^{2+3m} \to \mathbb{F}$ be defined as:

$$P_x(i,j,\vec{w}) = \sum_{i' \in \{0,\ldots,d'\}, j' \in [2m]} \delta_{i'}(i) \cdot \delta_{j'}(j) \cdot \hat{\alpha}_{i,j}(\vec{w}), \tag{4.4}$$

where $\delta_{i'}(i) = \prod_{a \in \{0,\ldots,d'\} \setminus \{i'\}} \frac{i-a}{i'-a}$ and $\delta_{j'}(j) = \prod_{a \in [2m] \setminus \{j'\}} \frac{j-a}{j'-a}$. Observe that for $i \in \{0,\ldots,d'\}$ and $j \in [2m]$ it holds that $P_x(i,j,\vec{w}) = \hat{\alpha}_{i,j}(x)$, but we emphasize that P_x is defined also for $i > d'$ and $j > 2m$. Since each $\hat{\alpha}_{i,j}$ has total degree $h \cdot \mathrm{polylog}(T)$, the total degree of P_x is at most $\Delta = d' + 2m + h \cdot \mathrm{polylog}(T) \leq h \cdot d \cdot \mathrm{polylog}(T) < 2p^{1/6} \cdot d \cdot \mathrm{polylog}(T)$, where the second inequality relies on the fact that $h \leq 2T^{\gamma/6}$ whereas $p \geq T^{\gamma \cdot c}$ (as specified in Proposition 4.2). Thus, a sufficiently large choice of $\bar{c}$ yields that $\Delta < \sqrt{p}$.

We use the polynomials P_x to define F as follows. The input to F is interpreted as $(x,\vec{z}) \in \{0,1\}^n \times \mathbb{F}^{2+3m}$ and we define $F(x,\vec{z}) = P_x(\vec{z})$. Observe that F still satisfies the properties that were asserted for L:

1. The reduction of $L^{(0)}$ to F maps input x to input $(x,(d',0,\vec{0}),0)$, and is thus a linear-time input-extending reduction. Indeed we have that $L^{(0)}(x) = \hat{\alpha}_{d',0}(\vec{0})_0 = F(x,(d',0,\vec{0}))_0$.

2. The function F is computable by logspace-uniform circuits of size $\mathrm{poly}(T)$ and depth $O(d \cdot \log(T)^4)$. This follows since, by Eq. (4.4),

$$F(x,(i,j,\vec{w})) = P_x(i,j,\vec{w}) = \sum_{i' \in \{0,\ldots,d'\}, j' \in [2m]} \delta_{i'}(i) \cdot \delta_{j'}(j) \cdot \hat{\alpha}_{i,j}(\vec{w}) \ , \tag{4.5}$$

   and the element $\hat{\alpha}_{i,j}(\vec{w})$ can be computed by $\log(|\mathbb{F}|)$ parallel calls to L (i.e., on inputs $L(x,(i,j),\vec{w},k)$ for $k = 1,\ldots,\log(|\mathbb{F}|)$), and we already showed that L is computable by a logspace-uniform circuit of size $\mathrm{poly}(T)$ and depth $O(d \cdot \log(T)^4)$.[32]

It is thus left to establish that F has a strongly tolerant same-length instance checker, as asserted in the following claim:

Claim 4.4.1. *For $\ell : \mathbb{N} \to \mathbb{N}$ such that $\ell(|(x,\vec{z})|) = |x|$ [33] it holds that F has an ℓ-strongly $(1/10)$-tolerant same-length instance checker running in time $\mathrm{poly}(d,n,\log(T))$.*

Proof. We use the reduction of F to L (described after Eq. (4.5)) and the instance checker M_L for L in order to construct a strongly tolerant instance checker M_F for F.

The algorithm M_F is given as input $(x,\vec{z}) \in \{0,1\}^n \times \mathbb{F}^{m'}$, where $m' = 2+3m$, and also has oracle access to some $A : \{0,1\}^n \times \mathbb{F}^{m'} \to \mathbb{F}$. For any $x \in \{0,1\}^n$, the oracle A determines a function $f_x : \mathbb{F}^{m'} \to \mathbb{F}$ such that $f_x(\vec{z}) = A(x,\vec{z})$.

The algorithm M_F interprets $\vec{z}$ as $(i,j,\vec{w})$, and for $k = 1,\ldots,\log(|\mathbb{F}|)$ it runs M_L on input $(x,(i,j),\vec{w},k)$, while answering its oracle queries as follows. Observe that every oracle query of M_L is of the form $(x,(i',j'),\vec{w'},k')$, for the same x as in the input to M_F. Given such a query, the algorithm M_F runs the decoder of Theorem 3.6 on the point $\vec{z'} = (i',j',\vec{w'})$, with soundness error $2^{-\mathrm{poly}(n)}$, relative to degree Δ, and using f_x as the alleged polynomial. The decoder returns an element $\alpha \in \mathbb{F}$ and M_F answers the query with the (k')-th bit of α. Assuming that M_L did not output $\perp$ in any of its invocations,

[32] We also rely on the fact that $\delta_{i'}$ and $\delta_{j'}$ can be computed by logspace-uniform circuits of size $\mathrm{polylog}(T)$ and depth less than $O(d \cdot \log(T))^4$, using the standard formula $\delta_i(\sigma') = \prod_{\sigma \in \{0,\ldots,d'\} \setminus \{i\}} \frac{\sigma - \sigma'}{i - \sigma'}$.

[33] That is, $\ell(n + (2+3m) \cdot \log(|\mathbb{F}|)) = n$.

the sequence of $\log(|\mathbb{F}|)$ output bits of the invocations of M_L defines a field element, which we denote by $\tilde{\alpha}_{i,j}(\vec{w})$, and the instance checker outputs

$$\sum_{i' \in \{0,\ldots,d'\}, j' \in [2m]} \delta_{i'}(i) \cdot \delta_{j'}(j) \cdot \tilde{\alpha}_{i,j}(\vec{w}) \ .$$

mimicking Eq. (4.5).

The running time of M_F is the running time of M_L multiplied by the running time $\mathrm{poly}(\Delta, m', \log(|\mathbb{F}|))$ of the decoder from Theorem 3.6, and is thus still bounded by $\mathrm{poly}(d, \log(T), n)$. The soundness of M_F follows immediately from the soundness of M_L. Thus we focus on establishing strongly tolerant completeness.

Fix $x \in \{0,1\}^n$ and $A \colon \{0,1\}^n \times \mathbb{F}^m \to \mathbb{F}$ such that $\Pr_{\vec{z} \in \mathbb{F}^m}[A(x,\vec{z}) \neq F(x,\vec{z})] \leq 1/10$. By the definitions of f_x and of F, we have that $\Pr_{\vec{z}}[f_x(\vec{z}) \neq P_x(\vec{z})] \leq 1/10$. By Theorem 3.6 and relying on the fact that $\Delta \leq \sqrt{p}$ and that p is prime, for each query $\vec{z}'$, the decoder outputs the value $P_x(\vec{z}')$ with probability at least $1 - 2^{-\mathrm{poly}(n)}$. Since the k'-th bit of $P_x(\vec{z}')$ is equal to $L(x, \vec{z}', k')$, with high probability all queries by M_L are answered according to L, in which case $\tilde{\alpha}_{i,j}(\vec{w}) = \hat{\alpha}_{i,j}(\vec{w})$ and M_F outputs $F(x,\vec{z})$. □

This concludes the proof of Proposition 4.4. ■

4.2 A reconstructive targeted somewhere-PRG

In this section we revisit the reconstructive targeted HSG from [CT21] and observe that the construction actually yields a *targeted somewhere PRG*; that is, the targeted generator outputs a collection of lists $W_1, \ldots, W_{d'}$, where d' is relatively small, and for every efficient algorithm D there exists $i \in [L]$ such that W_i is pseudorandom for D. We also show that the reconstruction R does not only compute the hard function at the given input x, but actually prints a small circuit whose truth-table is the value $f(x)$.

We will in fact prove something stronger. Instead of considering only a single algorithm D, we consider the more general setting in which there are d' sets of functions $F_1, \ldots, F_{d'}$, where each set F_i is associated with the corresponding list W_i. We show that for some $i \in [L]$, the generator is pseudorandom for *all* functions in the set F_i.

Theorem 4.5 (a reconstructive targeted somewhere PRG)**.** *There exists a universal constant $c > 1$ such that the following holds. Let $f \colon \{0,1\}^* \to \{0,1\}^*$ be computable by logspace-uniform circuits of size $T(n)$ and depth $d(n)$, let $\delta \colon \mathbb{N} \to (0,1)$, and let $M \colon \mathbb{N} \to \mathbb{N}$ such that*

$$c \cdot \log(T(n)) \leq M(n) \leq T(n)^{\delta(n)/c} \ ,$$
$$\delta(n) \geq c \cdot \frac{\mathrm{loglog}(T)}{\log(T)} \ .$$

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

1. **Generator.** *The generator G_f gets input x, runs in time $T^{O(1/\delta)}$, and outputs a collection of $d' = O(d \cdot \log(T)/\delta)$ lists of M-bit strings, denoted $W_1, \ldots, W_{d'}$. (We stress that the algorithm prints all of the strings in all of the lists "in a batch" in the stated time bound $T^{O(1/\delta)}$.)*

2. **Reconstruction.** *The reconstruction R gets input x and oracle access to a collection of d' sets of functions $\{0,1\}^M \to \{0,1\}$, denoted $F_1, ..., F_{d'}$, where each set F_i has at most L functions, and R runs in time $(d+n) \cdot L \cdot T^\delta$. Assume that for each $i \in [d']$, the set F_i contains a $(1/M)$-distinguisher for W_i. Then, with probability at least $1 - O(L \cdot \log(T)^2/T)$ the reconstruction R prints an oracle circuit C_f such that the truth-table of $C_f^{D_{d'}}$ is the string $f(x)$, where $D_{d'}$ is a function in $F_{d'}$.*

Proof. We follow the proof of [CT21, Theorem 5.1]. The proof first uses [CT21, Proposition 4.3] to argue that any f as in our hypothesis has suitable bootstrapping systems, and then uses [CT21, Proposition 4.4] to argue that any function f with such bootstrapping systems can be used to obtain a corresponding targeted HSG. Our goal is to prove two parts that were not included in the original statement:

1. We claim that R works when given d' sets of functions where for each $i \in [d']$ the set F_i contains a distinguisher for W_i (rather a single function that is a distinguisher for all the W_i's).

2. We claim that R prints (with high probability) an oracle circuit whose truth-table is $f(x)$ (rather than just printing $f(x)$).

To prove these statements, observe that the bootstrapping system obtained using [CT21, Proposition 4.3] with parameter $\mu = \delta/C$ (for a sufficiently large universal constant $C > 1$) has dimensions $d' \times T'$, where $T' = \text{poly}(T)$ and $d' = O(d \cdot \log(T)/\delta)$. [34] Now, in the proof of [CT21, Proposition 4.4], the generator enumerates over the d' layers of the bootstrapping system, encodes each layer P_i via the Hadamard code to $\bar{P}_i$, and uses the Nisan-Wigderson generator with $\bar{P}_i$ as the hard function (see [CT21, Theorem 4.8]) with parameters $(1^{\log(|\bar{P}_i|)}, 1^M, \Theta(\delta))$ to output a corresponding list of $T^{O(1/\delta)}$ strings of length M. We denote the set corresponding to the layer $\bar{P}_i$ by W_i, and note that the number of sets is d' as in our claim.

Let $\gamma = \delta/C'$ for a sufficiently large universal constant $C' > 1$. We now follow the proof of [CT21, Proposition 4.4], using precisely the same parameter values, the only change being that we change the reconstruction algorithm R as follows. The reconstruction R starts by constructing a circuit C_0 that computes the function whose truth-table is P_0 (i.e., the values of the input layer when padded to length T'), in time $\tilde{O}(\max\{n, T^\mu\} \cdot T^\mu)$. We will then rely on the following lemma:

Lemma 4.5.1 (a strengthening of [CT21, Lemma 4.10])**.** *There is a universal constant c_0 and an algorithm A_{step} that gets as input an oracle circuit C_{i-1} and oracle access to D_{i-1} such that the truth-table of $C_{i-1}^{D_{i-1}}$ is P_{i-1}, and also gets oracle access to a function $D_i \colon \{0,1\}^M \to \{0,1\}$, runs in time $T^{2(\gamma+\mu)} \cdot (M \cdot |C_{i-1}|)^{c_0}$, and with probability at least $1 - 1/T^2 - 4 \cdot 2^{-M}$ satisfies the following:*

1. *If D_i is a $(1/M)$-distinguisher for W_i, then A_{step} prints an oracle circuit C_i of size $M^{c_0} \cdot T^{2(\gamma+\mu)}$ such that the truth-table of $C_i^{D_i}$ is P_i.*

2. *For every D_i, the algorithm A_{step} either prints an oracle circuit C_i as in Item (1), or outputs $\perp$*

Proof. In the original statement of [CT21, Lemma 4.10], both oracles D_{i-1} and D_i are the same function D, and this single function is a distinguisher for all the lists that the

[34] The original statement asserts that $d' = O(d \cdot \log(T))$, but the proof yields the value $d' = O(d \cdot \log(T)/\mu) = O(d \cdot \log(T)/\delta)$.

generator outputs. However, the proof itself only relies on the hypotheses that $C_{i-1}^{D_{i-1}}$ computes the function whose truth-table is P_{i-1}, and that D_i is a $(1/M)$- distinguisher for S_i. This establishes Item (1).

To see what happens when D_i is not a $(1/M)$-distinguisher for W_i, recall that the first step of the original algorithm A is to run the Nisan-Wigderson reconstruction, hoping to obtain a circuit $C_{i,1}$ such that $C_{i,1}^{D_i}$ computes P_i correctly on $1/2 + M^{-3}$ of the inputs (see [CT21, Claim 4.10.1 and Theorem 4.8]). Our modified algorithm does the same thing, but after this step it also *tests* that $C_{i,1}^{D_i}$ computes P_i correctly on $1/2 + M^{-3}$ of the inputs. That is, it samples $O(M^{11})$ random inputs in $[T']$, evaluates $C_{i,1}^{D_i}$ and $C_{i-1}^{D_{i-1}}$ on these inputs, and proceeds if and only if the two latter functions agree on at least $1/2 + M^{-3} - M^{-5}$ of the sample (otherwise it outputs $\perp$). Then it proceeds as in the original proof, replacing the guarantee that $\Pr_{z\in[T']}[C_{i-1}^{D_{i-1}}(z) = P_i(z)] \geq 1/2 + M^{-3}$ by a guarantee that $\Pr_{z\in[T']}[C_{i-1}^{D_{i-1}}(z) = P_i(z)] \geq 1/2 + M^{-4}$; the original proof is already insensitive to the difference between M^{-3} and M^{-4}.

The original success probability was $1 - 1/T^2 - 3 \cdot 2^{-M}$, and the new success probability is lower by an additive factor of 2^{-M} (due to the probability that the sampling above is incorrect). The original running time was $T^{2(\gamma+\mu)} \cdot \mathrm{poly}(M, |C_{i-1}|)$, and it increases by an additive factor of $M^{11} \cdot \tilde{O}(|C_{i-1}| + |C_{i,1}|) < \mathrm{poly}(M, |C_{i-1}|) \cdot T^{2\gamma}$, where the inequality is since $M \geq \log(T)$ and the size of $C_{i,1}$ is at most $T^{2\gamma} \cdot \mathrm{poly}(M)$ (see [CT21, Claim 4.10.1] for the bound on $|C_{i,1}|$). □

The reconstruction algorithm R works in d' iterations. In each iteration $i \in [d']$ it starts with a circuit C_{i-1} such that $C_{i-1}^{D_{i-1}}$ computes P_{i-1}, where D_{i-1} is one of the oracles in F_{i-1} (when $i = 1$ no oracle is needed, so we think of D_0 as a trivial function). Then, for each $j = 1, ..., L$, the reconstruction runs the algorithm A_{step} from Lemma 4.5.1 with oracle $D_j^{(i)}$. If A_{step} outputs an oracle circuit C_i , the algorithm proceeds to iteration $i + 1$, with circuit C_i and oracle $D_i = D_j^{(i)}$; otherwise (i.e., if A_{step} outputs $\perp$), the reconstruction continues in iteration i with oracle $D_{j+1}^{(i)}$. If A_{step} outputs $\perp$ for all $j \in [L]$, the reconstruction aborts and outputs $\perp$.

By a union-bound, the probability that all $\leq d' \cdot L$ invocations of Lemma 4.5.1 are successful is at least

$$\begin{aligned} 1 - d' \cdot L \cdot (T^{-2} + 4 \cdot 2^{-M}) &\geq 1 - L \cdot \frac{2d \cdot \log(T)}{\delta} \cdot T^{-2} && (M > c \cdot \log(T)) \\ &\geq 1 - O(L \cdot \log(T)^2/T) \,. && (d \leq T) \end{aligned}$$

In this case, by our hypothesis that for each $i \in [d']$ the set F_i contains a distinguisher for W_i, after iteration d' we obtained $D_{d'} \in F_{d'}$ and $C_{d'}$ such that the truth-table of $C_{d'}^{D_{d'}}$ is the string $\bar{P}_{d'}$. The latter string is the Hadamard encoding of $P_{d'}$, and by the construction of $P_{d'}$ in [CT21, Proof of Proposition 4.3], the prefix of length $|f(x)|$ in $P_{d'}$ is the string $f(x)$. Thus, the reconstruction algorithm can output an oracle circuit that gets $y \in [|f(x)|]$, computes the index z in the Hadamard encoding $\bar{P}_{d'}$ of $P_{d'}$ such that $\bar{P}_{d'}(z) = P_{d'}(y)$, and outputs $C_{d'}^{D_{d'}}(z) = \bar{P}_i(z) = P_i(y) = f(x)_y$.

The running time of the reconstruction algorithm is

$$\begin{aligned}
&\tilde{O}(\max\{n, T^{\mu}\} \cdot T^{\mu}) + d' \cdot L \cdot T^{2(\gamma+\mu)} \cdot (M \cdot |C_{i-1}|)^{c_0} \\
&< n \cdot T^{2\mu} + d' \cdot L \cdot T^{2(\gamma+\mu)} \cdot (M^{c_0} \cdot T^{2(\gamma+\mu)})^{c_0} && (|C_{i-1}| \leq M^{c_0} \cdot T^{2(\gamma+\mu)}) \\
&< (d+n) \cdot T^{\delta/2} \cdot L \cdot M^{c_0^2} && (\mu = \delta/C, \gamma = \delta/C') \\
&< (d+n) \cdot T^{\delta} \cdot L\,, && (M < T^{\delta/2c_0^2})
\end{aligned}$$

for a sufficiently large universal constant $c > 2c_0^2$ in the hypothesis $M \leq T^{\delta/c}$. ■

5 High-end hardness: Breaking the $\mathcal{PSPACE}$ barrier

In this section we prove the results from Section 1.1. Specifically, in Section 5.1 we prove the general version of Theorem 1.1, and in Section 5.2 we prove Theorem 1.2.

5.1 High-end hardness-to-randomness tradeoff

We now prove Theorem 1.1, which asserts a hardness vs randomness tradeoff that is based on hardness in a class that is (probably) larger than $\mathcal{PSPACE}$, and that can yield polynomial time derandomization on all input lengths. In fact, we prove the results for general time bounds that satisfy some natural conditions.

Definition 5.1 (nice time bounds)**.** *We say that $T\colon \mathbb{N} \to \mathbb{N}$ is a* nice time bound *if T is non-decreasing and time-computable and satisfies:*

1. *For every sufficiently large $n \in \mathbb{N}$ it holds that $T(n+1) \leq 2T(n) \leq 2^n$.*
2. *For every two constants $\alpha, \mu \in (0,1)$ such that $\alpha \geq \mu$ and every sufficiently large $n \in \mathbb{N}$ it holds that $T(\alpha \cdot n) \geq T(n)^{\mu}$.*
3. *For any constant $\alpha \in (0,1)$ there exists $c > 1$ such that given any integer $n \in \mathbb{N}$, we can compute an integer ℓ such that $T(\alpha \cdot \ell) \in [n, c \cdot n]$ in space $O(\ell)$.*

The time bounds typically considered in complexity theory are nice; for example, $T(n) \in \left\{n^k, 2^{(\log n)^k}, 2^{n^{\epsilon}}, 2^{\epsilon \cdot n}\right\}$ for constants $k \geq 1$ and $\epsilon \in (0,1)$ are all nice.

We first prove the derandomization of $\mathcal{RP}$, and then prove the derandomization of $\mathcal{BPP}$ with small advice. The first proof uses the targeted somewhere-PRG from Theorem 4.5 along with the instance checker from Proposition 4.4.

Theorem 5.2 (the general version of Theorem 1.1 for $\mathcal{RP}$)**.** *For any nice time bound $T\colon \mathbb{N} \to \mathbb{N}$ that is larger than a sufficiently large polynomial, and any $d(n) \leq T(n)^{\delta}$, where $\delta > 0$ is a universal constant, the following holds. Assume that $\mathsf{lu}\text{-}\mathcal{CKT}[2^{O(n)}, d(n)] \not\subseteq \mathtt{i.o.}\mathcal{BPTIME}[T]$. Then, for every constant $a \in \mathbb{N}$ we have that*

$$\mathcal{RP} \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{heur}_{1-1/n^a}\text{-}\mathcal{DTIME}[2^{t_c(n)}]\,,$$

where $t_c(n) = (T^{-1}(n^c))^2/\log(n)$. [35]

[35] The notation $T^{-1}(n^c)$ means the largest integer ℓ such that $T(\ell) \leq n^c$.

To see that the first part of the conclusion of Theorem 1.1 follows from Theorem 5.2, instantiate the latter with $T(n) = 2^{\epsilon \cdot n}$ and $d(n) = 2^{\epsilon \cdot \delta \cdot n}$, in which case $t_c(n) = O_c(\log(n))$.

Proof of Theorem 5.2. Let $L^{(0)} \in \text{lu-}\mathcal{CKT}[2^{O(n)}, d(n)] \setminus \text{i.o.}\mathcal{BPTIME}[T]$ and let $L^{(1)} \in \text{lu-}\mathcal{CKT}[2^{O(n)}, O(d \cdot n^4)]$ be the corresponding instance-checkable problem from Proposition 4.4, and recall that the reduction maps inputs of length n to inputs of length at most $c' \cdot n$, where $c' > 1$ is a universal constant and we relied on the bound $T(n) \leq 2^n$. By Lemma A.1 there exists $L' \in \text{lu-}\mathcal{CKT}[2^{O(n)}, O(d \cdot n^4)] \setminus \text{i.o.}\mathcal{BPTIME}[T']$ that is also same-length instance-checkable in time $\text{poly}(d, \log(T), n)$, where

$$T'(n) = \sqrt{T(\alpha \cdot n)} \leq \frac{T(n/2c')}{\text{poly}(d, n, \log(T))} ,$$

and $\alpha = 1/2c'$ and the inequality relies on our assumptions that $d \leq T^\delta$ for a universal $\delta \ll \alpha/2$ and that T is nice and is larger than a sufficiently large polynomial.

Let $L \in \mathcal{RTIME}[n^k]$, let $a \in \mathbb{N}$, let A be a probabilistic algorithm solving L in time n^k, and let S be a probabilistic algorithm sampling $\mathbf{x}$ in time n^k. Fix a sufficiently large constant c that depends on a and on k and on $L^{(0)}$.

The derandomization algorithm. Since T is nice, for any integer of the form n^{2c}, we can compute an integer $\ell = \ell(n)$ such that $T(\alpha \cdot \ell) \in [n^{2c}, O(n^{2c})]$ in space $O(\ell)$. By the definition of $T'(\ell) = \sqrt{T(\alpha \cdot \ell)}$, we have that

$$n^c \leq T'(\ell) \leq O(n^c) ,$$

and since we defined $T^{-1}(N) = \max\{\ell' : T(\ell') \leq N\}$, we have

$$\ell \leq T^{-1}(O(n^{2c})) . \tag{5.1}$$

Consider the function $f\colon \{0,1\}^n \to \{0,1\}^{2^\ell}$ that maps any n-bit input to the truth-table of $L'_\ell = L' \cap \{0,1\}^\ell$. Note that f is computable by logspace-uniform circuits of size $S = 2^{O(\ell)}$ and depth $d' = d(\ell) \cdot \ell^4$, relying on our hypothesis about L' (and relying on the fact that ℓ can be computed in space $O(\ell) = O(\log(S))$).

Now, let G_f be the targeted somewhere-PRG from Theorem 4.5, instantiated with the function f and with parameters $\delta = \Theta(\log(T'(\ell)/\ell))$ and $M = n^k$. [36] Note that G_f runs in time

$$\begin{aligned} 2^{O(\ell/\delta)} &= 2^{O(\ell^2/\log(T'(\ell)))} \\ &= 2^{O(\ell^2/\log(n))} && (T'(\ell) = \Theta(n^c)) \\ &\leq 2^{O(t_{3c}(n))} , \end{aligned}$$

where the last inequality relied on the fact that $t_{3c}(n) = T^{-1}(n^{3c})^2/\log(n) \geq \ell^2/\log(n)$ (by Eq. (5.1). The output of G_f is K lists of M-bit strings, denoted $W_1, ..., W_K$, where

$$K = O(d' \cdot \log(S)/\delta) = O(d(\ell) \cdot \ell^6/\log(T'(\ell))) .$$

[36] Theorem 4.5 relies on the hypotheses $\delta \geq c_0 \cdot \text{loglog}(S)/\log(S)$ and $c_0 \cdot \log(S) \leq M \leq S^{\delta/c_0}$ $\iff$ $\Theta(\ell) \leq n^k \leq 2^{\Theta(\log(n^c))}$, for a universal constant c_0. These hypotheses are satisfied by the assumption that T is larger than a sufficiently large polynomial and by a sufficiently large choice of c.

Given input $x \in \{0,1\}^n$, we invoke G_f as above to obtain the lists, run $A(x,w)$ for each $w \in \bigcup_{i\in[K]} W_i \subseteq \{0,1\}^M$, and output 1 if and only if there exists w such that $A(x,w) = 1$. Note that the running time of this algorithm is

$$2^{O(t_{3c}(n))} + O(K \cdot n^k) < 2^{O(t_{3c}(n))} \cdot n^k < 2^{O(t_{3c}(n))} ,$$

where the last inequality is since $T(\ell) \leq 2^\ell$.

The reconstruction argument. Assume towards a contradiction that for infinitely many $n \in \mathbb{N}$, when running S on input 1^n, with probability at least $1/n^a$ over its random coins it outputs $x \in \{0,1\}^n$ such that $\Pr_{r\in\{0,1\}^{n^c}}[A(x,r) = 1] \geq 1/2$ but $A(x,w) = 0$ for every $w \in \bigcup_{i\in[K]} W_i$.

In this case, we can compute L'_ℓ as follows. Given $z \in \{0,1\}^\ell$, run $S(1^n)$ for $h = O(n^a)$ times to obtain a list of strings $x_1, ..., x_h \in \{0,1\}^n$. For each $i \in [h]$,

1. We run the instance checker for L' with input z, and whenever the instance checker queries a location $q \in \{0,1\}^\ell$ we answer as follows.

2. We run the reconstruction algorithm R from Theorem 4.5 on input q, while providing it oracle access to the function $D_{x_i}(r) = A(x_i, r)$. (In Theorem 4.5, the reconstruction R is allowed oracle access to several sets of functions. In the current instantiation, all sets are identical and consist of the single function D_{x_i}.) If the algorithm returns a circuit C we answer the query by $C^{D_{x_i}}(q)$, and otherwise we halt the entire procedure and output $\perp$.

3. We denote the value that the instance checker returns by $v_i \in \{0,1,\perp\}$.

If there exists $\sigma \in \{0,1\}$ such that for some $i \in [h]$ it holds that $v_i = \sigma$ and for all $i \in [h]$ it holds that $v_i \in \{\sigma, \perp\}$, we return σ; otherwise we return zero.

Now, by our assumption about S, with high probability over the random coins of S there exists $i \in [h]$ such that $\Pr_r[A(x_i,r) = 1] \geq 1/2$ but $A(x,w) = 0$ for every $w \in \bigcup_{i\in[K]} W_i$, and we condition on this event. Also, we can assume that the error probability of the instance checker and of R is at most $2^{-n^{2a}}$ (by using naive error-reduction with $O(n^{2a})$ repetitions[37]). Thus, with high probability, all the invocations of R and of the instance checker are successful, and we condition on this event too. In this case we have that $v_i = L_{\ell'}(z)$ whereas for all $j \neq i$ it holds that $v_j \in \{L'_\ell(z), \perp\}$, and therefore the algorithm correctly outputs $L'_\ell(z)$.

Finally, the running time of the algorithm is

$$\underbrace{O(n^{a+k})}_{\text{sampler } S} \cdot \underbrace{n^{2a} \cdot \text{poly}(d', \ell)}_{\text{instance checker}} \cdot \underbrace{(d' + \ell^{c_{4.5}}) \cdot T^\delta \cdot n^{c_{4.5}+k+2a}}_{\text{reconstruction } R} \leq \text{poly}(d') \cdot T^\delta \cdot n^{c/2} < T'(\ell) ,$$

where $c_{4.5}$ is the universal constant from Theorem 4.5, and the first inequality is by the assumption that c is sufficiently large, and the second inequality is by the assumption that $\delta > 0$ is sufficiently small. This contradicts the hypothesized hardness of L. ∎

We now prove the derandomization of $\mathcal{BPP}$ with small advice. In the following statement, the hypothesis is identical to that of Theorem 5.2, and only the conclusion

[37] To reduce the error of R, given query q we run R for $O(n^{2a})$ times, and whenever R outputs a circuit C we record the answer $C(q)$. We output the majority vote among the recorded answers.

is different. In the proof, instead of using instance checkers, we will use the more refined guarantee in Theorem 4.5 that allows the reconstruction algorithm to get access to several sets of potential distinguishers.

Theorem 5.3 (the general version of Theorem 1.1 for $\mathcal{BPP}$)**.** *Under the same hypothesis of Theorem 5.2, for every constant $a \in \mathbb{N}$ we have that*

$$\mathcal{BPP} \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{heur}_{1-1/n^a}\text{-}\mathcal{DTIME}[2^{t_c(n)}]/\alpha_c(n) \ ,$$

where $t_c(n) = (T^{-1}(n^c))^2/\log(n)$ and $\alpha_c(n) = \log(d(T^{-1}(n^c))) + O(\log(T^{-1}(n)))$.

To see that the second part of the conclusion of Theorem 1.1 follows from Theorem 5.3, observe that when $T(n) = 2^{\epsilon \cdot n}$ and $d(n) = 2^{\epsilon \cdot \delta \cdot n}$ we have $t_c(n) = O_c(\log(n))$ and $\alpha_c = O_c(\log(n))$.

Proof of Theorem 5.3. We instantiate the targeted somewhere-PRG from Theorem 4.5 with the exact same function and paramters as in the proof of Theorem 5.2, but now the generator also receives advice $i \in [K]$, and prints only the strings in the i^{th} output list W_i. On input x we output $\mathsf{MAJ}_{w \in W_i}\{A(x,w)\}$.

To prove correctness, assume towards a contradiction that for every advice string $i \in [K]$, the probability over $x \leftarrow S(1^n)$ (i.e., over an n-bit input chosen by the sampler S) that $\Pr_{r \in \{0,1\}^{n^k}}[A(x,r)=1] \notin \Pr_{w \in W_i}[A(x,w)] \pm 1/10$ is at least $1/n^a$. We show below a reconstruction procedure whose complexity can be bounded in a way that is identical to the one in the proof of Theorem 5.2, but that works with the current relaxed assumption. It follows that there exists a fixed $i \in [K]$ (that may depend on L and on S) such that the probability that S samples x for which $\Pr_{r \in \{0,1\}^{n^k}}[A(x,r)=1] \notin \Pr_{w \in W_i}[A(x,w)] \pm 1/10$ is at most $1/n^a$. The advice length is

$$\lceil \log(K) \rceil = \log(d') + 2\log(\ell) - \mathrm{loglog}(n) + O(1) = \log(d(\ell)) + O(\log(\ell)) \ ,$$

and by Eq. (5.1) this is at most $\alpha_{3c}(n)$.

Turning to the reconstruction argument, given input $z \in \{0,1\}^\ell$ for L'_ℓ, we sample $h' = O(n^a \cdot \log(K) \cdot K)$ strings in $\{0,1\}^n$ and partition them into K sets $X_1, ..., X_K$ of h'/K strings each. We then run the reconstruction algorithm R from Theorem 4.5 with input z, giving it access to the K sets of functions defined for each $i \in [K]$ by $F_i = \{D_{x_i}\}_{x_i \in X_i}$ where $D_{x_i}(r) = M(x,r)$. If R prints a circuit C we output $C(z)$, and otherwise we output some default value, say 0. By our assumption, and using a union-bound over $i \in [K]$, with probability at least 0.99, for every $i \in [K]$ there exists $x_i \in X_i$ such that D_{x_i} is a $(1/10)$-distinguisher for W_i. In this case, with probability $1 - o(1)$ we have that $C(z) = L'_\ell(z)$. The running time of this procedure is at most

$$\begin{aligned} & h' \cdot n^k + (d' + n) \cdot O(n^a \cdot \log(K)) \cdot T^\delta \\ & \leq n^{a+k} \cdot \mathrm{poly}(d', \ell) \cdot T^\delta && (K \leq \mathrm{poly}(d', \ell)) \\ & \leq \mathrm{poly}(d') \cdot T^\delta \cdot n^{c/2} \ , \end{aligned}$$

which is at most $T'(\ell)$, contradicting the hardness of L'. ∎

5.2 Optimal worst-case to average-case reductions

In this section we prove Theorem 1.2, which asserts an optimal worst-case to average-case reduction for computing problems in $\mathcal{SPACE}[O(n)]$ by probabilistic algorithms. Its proof uses the instance checker from Proposition 4.2.

Theorem 5.4 (optimal worst-case to average-case reduction for linear space; Theorem 1.2, restated)**.** *For every non-increasing function $\epsilon\colon \mathbb{N} \to (0,1)$ such that $\epsilon(n) \geq 2^{-n}$ is computable in space $O(\log(n))$ there is a language $\hat{L}$ that is complete for $\mathcal{SPACE}[O(n)]$ under linear-time reductions such that*

$$\mathcal{SPACE}[O(n)] \not\subset \texttt{i.o.}\mathcal{BPTIME}[T] \implies \hat{L} \notin \texttt{i.o.-avg}_{1/2+\delta}\text{-}\mathcal{BPTIME}[T'] \,,$$

where $T'(n) = T(n/c)/(n/\epsilon(\lfloor n/c' \rfloor))^c$ and $\delta(n) = \epsilon(\lfloor n/c' \rfloor)$, for a constant $c > 1$ that depends on the hypothesized lower bound and a universal constant $c' > 1$.

Proof. Let L be the linear-space complete problem from Proposition 4.3 that is same-length instance checkable in polynomial time. Let $\hat{L}$ be the problem obtained by encoding, for every $n \in \mathbb{N}$, the truth-table of $L \cap \{0,1\}^n$ via the error-correcting code Enc from Theorem 3.8, instantiated with the function $\epsilon(n)$. Since the code is computable in space that is logarithmic in its output length, the problem $\hat{L}$ is in linear space; and since $\hat{L}$ is systematic, it holds that L is reducible to $\hat{L}$ in linear time.

Note that $\hat{L}$ was so far defined only on input lengths in $S = \{c_0 \cdot n_0\}_{n_0 \in \mathbb{N}}$, where c_0 is the constant from the linear-time reduction of L to $\hat{L}$. [38] For every $n \notin S$, we define $\hat{L}_n$ such that $\hat{L}_n(x) = 1$ if and only if $\hat{L}_m(x_1, ..., x_m) = 1$, where $m = c_0 \cdot \lfloor n/c_0 \rfloor$. (Using the terminology from Definition 3.3, the problem $\hat{L}$ is the natural S-extension of the naive target problem of the reduction.) This problem is still complete for $\mathcal{SPACE}[O(n)]$.

Now, assume that there exists

$$L^{(0)} \in \mathcal{SPACE}[O(n)] \setminus \texttt{i.o.}\mathcal{BPTIME}[T] \,.$$

By Lemma A.1, we have that

$$\bar{L} \notin \texttt{i.o.}\mathcal{BPTIME}[T'] \,,$$

where $\bar{L}$ is the natural extension of L as in Definition 3.3 and $T'(n) = T(n/c')/n^{c'}$ and $c' > 1$ depends on $L^{(0)}$ (specifically, on the precise linear time complexity of reducing $L^{(0)}$ to L). Then, by Lemma A.4, we have that

$$\hat{L} \notin \texttt{i.o.-avg}_{1/2+\delta}\text{-}\mathcal{BPTIME}[T''] \,,$$

where $\delta(n) \geq \epsilon(\lfloor n/2c_0 \rfloor)$ and

$$T''(n) \geq \frac{T'(n/2c_0)}{(n/\epsilon(\lfloor n/2c_0 \rfloor))^{c''}} = \frac{T(n/(2c_0 \cdot c'))}{(n/\epsilon(\lfloor n/2c_0 \rfloor))^{c' \cdot c''}}$$

and $c'' > 1$ is universal. ∎

[38] The reduction maps inputs of length n_0 to inputs of length $c \cdot (n_0 + \log(1/\epsilon(n_0))) \leq 2c \cdot n_0$, where the inequality is since $\epsilon(n_0) \geq 2^{-n_0}$ and c is the constant hidden in the definition of $\bar{N}$ in Theorem 3.8.

6 Mild average-case hardness in a subclass of $\mathcal{P}$

In this section we prove Theorem 1.3. Recall that in this result we start from mild average-case hardness in a large subclass of $\mathcal{P}$, and deduce polynomial-time average-case derandomization of $\mathcal{RP}$. Indeed, the assumption refers to hardness with very little "structure" (in the sense that an arbitrary function in a large subclass of $\mathcal{P}$ will do), but requires mild average-case hardness rather than worst-case hardness.

Theorem 6.1 (derandomization from mild average-case hardness in a subclass of $\mathcal{P}$)**.** *For every four constants $a, b, c_1, c_2 \in \mathbb{N}$ there exists a constant $c_3 > 1$ such that following holds. Assume that there exists a language*

$$L^{(0)} \in \text{lu-}\mathcal{CKT}[\text{poly}(n), n^{c_1}] \setminus \text{i.o.avg}_{(1-n^{-c_2})}\text{-}\mathcal{BPTIME}[n^{c_3}] \,.$$

Then, $\mathcal{RTIME}[n^b] \subseteq \text{avg}_{(1-n^{-a})}\text{-}\mathcal{P}$.

Proof. Let $a, b, c_1, c_2 \in \mathbb{N}$ be constants and let c_3 be a sufficiently large constant to be determined below. Let

$$L^{(0)} \in \text{lu-}\mathcal{CKT}[\text{poly}(n), n^{c_1}] \setminus \text{i.o.-avg}_{(1-n^{-c_2})}\text{-}\mathcal{BPTIME}[n^{c_3}] \,,$$

and let F be the corresponding function from the "furthermore" part of Proposition 4.4.

Reduction to an instance-checkable problem. Recall that F has an ℓ-strongly $(1/10)$-tolerant same-length instance checker running in time $n^{c \cdot c_1}$, for some universal constant $c > 1$, and that $L^{(0)}$ is reducible to F via an input-extending reduction that maps inputs of length n to inputs of length $\ell^{-1}(n) = O(n)$. We rely on the following lemma:

Lemma 6.2 (preserving mild average-case hardness on all input lengths under reductions using a strongly tolerant instance checker)**.** *Let $L^{(0)} \notin \text{i.o.-avg}_{(1-n^{-c_2})}\text{-}\mathcal{BPTIME}[n^{c_3}]$, and let $F\colon \{0,1\}^* \to \{0,1\}^*$ such that for some constant $c' > 1$:*

1. *Deciding $L^{(0)}$ is reducible in linear time to computing F via an input-extending reduction that maps inputs of length n to inputs of length $c' \cdot n$.*
2. *The function F has an ℓ-strongly $(1/10)$-tolerant same-length instance checker with running time $n^{c \cdot c_1}$, where $\ell(n) = n/c'$.*

Then, the natural S-extension of F is not in $\text{i.o.-avg}_{(1-n^{-c_2}/100)}\text{-}\mathcal{BPTIME}[n^{c_3 - c \cdot c_1 - 1}]$, where $S = \{c' \cdot n_0 : n_0 \in \mathbb{N}\}$.

We defer the proof of the lemma to Appendix A (see Lemma A.2). Relying on the lemma, it follows that for $S = \{O(n_0) : n_0 \in \mathbb{N}\}$, the natural S-extension of F, denoted $\bar{F}$, is not in $\text{i.o.-avg}_{(1-n^{-c_2}/100)}\text{-}\mathcal{BPTIME}[n^{c_5 - c \cdot c_1 - 1}]$. Using the upper-bound on F from Proposition 4.4, we have that

$$\bar{F} \in \text{lu-}\mathcal{CKT}[\text{poly}(n), n^{c_1+1}] \setminus \text{i.o.-avg}_{(1-n^{-(c_2+1)})}\text{-}\mathcal{BPTIME}[n^{c_3'}] \,, \tag{6.1}$$

where $c_3' = c_3 - c \cdot c_1 - 1$.

Hardness amplification via direct-product. Now, for $k(n) = \Theta(\log(n^{a \cdot c_2}) \cdot n^{3c_2})$, let $G = \bar{F}^{\otimes k}$ be the k-wise direct-product function of $\bar{F}$, and let $\bar{G}$ be the natural S_k-extension of G, where $S_k = \{k \cdot n\}_{n \in \mathbb{N}}$. By Fact 3.14, for every $\delta = \delta(n) > 0$ it holds that $\bar{F}$ has a $(\delta, 10\delta)$-tolerant instance checker running in time $n^{c \cdot c_1}$. We rely on the following lemma:

Lemma 6.3 (amplifying average-case hardness on all input lengths using a direct product and an instance checker). *Let $\bar{F} \notin \texttt{i.o.-avg}_{(1-n^{-(c_2+1)})}\text{-}\mathcal{BPTIME}[n^{c_3'}]$ that has a $(n^{-3(c_2+1)}, 10n^{-(c_2+1)})$-tolerant same-length instance checker with running time $n^{c \cdot c_1}$, and let $G = \bar{F}^{\otimes k}$ be the k-wise direct product of $\bar{F}$, for $k = \Theta(\log(n^{a \cdot c_2}) \cdot n^{3c_2})$. Then, for $S_k = \{k \cdot n_0 : n_0 \in N\}$, the natural S_k-extension of G is not in $\texttt{i.o.-avg}_{n^{-a}}\text{-}\mathcal{BPTIME}[n^{c_3''}]$, where $c_3'' = \alpha \cdot \frac{c_3'}{c \cdot c_1 \cdot (c_2+1) \cdot a}$ for a universal constant $\alpha > 0$.*

We defer the proof of the lemma to Appendix A (see Lemma A.5). Relying on the lemma and on the naive upper-bound for computing $\bar{G}$ (by computing $\bar{F}$ independently on each of the k instances in parallel), we have that

$$\bar{G} \in \text{lu-}\mathcal{CKT}[\text{poly}(n), n^{c_1+1}] \setminus \texttt{i.o.-avg}_{n^{-a}}\text{-}\mathcal{BPTIME}[n^{c_3''}] \,, \tag{6.2}$$

where $c_3'' = \alpha \cdot \frac{c_3 - c \cdot c_1 - 1}{c \cdot c_1 \cdot (c_2+1) \cdot a}$. (Note that the upper bound in Eq. (6.2) is wasteful, since the depth is actually m^{c_1+1} where $m = n/k$, rather than n^{c_1+1}. This does not affect our analysis, so we do not optimize it.)

The derandomization. Let $L \in \mathcal{RTIME}[n^b]$, and let A be a probabilistic algorithm that solves L with one-sided error in time n^b. We instantiate Theorem 4.5 using $\bar{G}$ as the hard function with parameters $M(n) = n^b$ and $T = \text{poly}(n)$ and $d = n^{c_1+1}$ and sufficiently small constant $\delta > 0$. [39] Given input $x \in \{0,1\}^n$, we simulate A using the pseudorandom strings that the generator from Theorem 4.5 prints, and accept if and only if A accepts with at least one pseudorandom string. The running time of this algorithm is $\text{poly}(n)$.

Denote the deterministic algorithm above by D, and assume towards a contradiction that for infinitely many $n \in \mathbb{N}$ it holds that $\Pr_{x \in \{0,1\}^n}[D(x) \neq L(x)] \geq n^{-a}$. Consider the algorithm R from Theorem 4.5 when given D as an oracle (i.e., all the sets of oracle functions for R are the singleton $\{D\}$). In this case, the algorithm R^D runs in time at most

$$(d+n) \cdot T^{\delta} \leq n^{2c_1 \cdot b} \,,$$

where the last inequality is by a sufficiently small choice of δ, and R^D decides L correctly on a n^{-c_3} fraction of the inputs. We reach a contradiction by setting c_3 to be sufficiently large such that $c_3'' > 2c_1 \cdot b$. [40] ∎

Since a large part of the proof above will be reused in Section 7, let us now state this part as a stand-alone. The proof above started with a language $L^{(0)}$ that is mildly hard on average, obtained an instance-checkable function $\bar{F}$ that is mildly hard on average, and used $\bar{F}$ (coupled with hardness amplification) for derandomization. The following claim abstracts the second step of using $\bar{F}$ for derandomization:

[39] We ensure that the hypothesis $M \leq T^{\delta/c}$ is met by assuming (wlog) that T is a sufficiently large polynomial and taking $\delta > 0$ to be sufficiently small.

[40] That is, we set $c_3 > (2c/\alpha) \cdot (c_1)^2 \cdot (c_2+1) \cdot a \cdot b + c \cdot c_1 + 1$.

Proposition 6.4 (from an instance-checkable function that is mildly hard on average to derandomization)**.** *For every five constants $a, b, c_1, c_2, c_3 \in \mathbb{N}$ there exists a sufficiently large constant $c_4 > 1$ such that the following holds. Assume that there exists a function*

$$\bar{F} \in \mathsf{lu}\text{-}\mathcal{CKT}[\mathrm{poly}(n), n^{c_1}] \setminus \mathtt{i.o.}\text{-}\mathsf{avg}_{1-n^{-c_2}}\text{-}\mathcal{BPTIME}[n^{c_4}] \,,$$

such that $\bar{F}$ has an ℓ-strongly tolerant same-length instance checker running in time n^{c_3}, for some $\ell(n)$. Then, $\mathcal{RTIME}[n^b] \subseteq \mathsf{avg}_{1-n^{-a}}\text{-}\mathcal{P}$.

The proof of Proposition 6.4 is identical to the part of the proof of Theorem 6.1 that starts from Eq. (6.1).

7 Hardness for low-degree arithmetic formulas

In this section we prove Theorem 1.5, which asserts that worst-case hardness of a function computable by low-degree arithmetic formulas of polynomial size implies polynomial-time derandomization of $\mathcal{RP}$, on average. Since the proof refers to the precise details of this computational model, let us begin by defining arithmetic formulas and establishing some conventions.

Definition 7.1 (arithmetic formulas)**.** *An arithmetic formula ϕ on n variables $x_1, \ldots, x_n$ of size $\mathcal{SIZE}(\phi)$ and degree $\deg(\phi)$ is a syntactic model defined recursively by one of the following operations:*

1. *$\phi = x_i$ for $i \in [n]$, or $\phi = \sigma$ for a constant $\sigma \in \{0,1\}$, in which case $\mathcal{SIZE}(\phi) = 1$ and $\deg(\phi) = \begin{cases} 1 & \phi = x_i \\ 0 & \phi \in \{0,1\} \end{cases}$.*
2. *$\phi = -\psi$, where ψ is an arithmetic formula over $x_1, \ldots, x_n$, in which case $\mathcal{SIZE}(\phi) = \mathcal{SIZE}(\psi) + 1$ and $\deg(\phi) = \deg(\psi)$.*
3. *$\phi = \psi_1 + \psi_2$, where both of ψ_1, ψ_2 are arithmetic formulas over $x_1, \ldots, x_n$, in which case $\mathcal{SIZE}(\phi) = \mathcal{SIZE}(\psi_1) + \mathcal{SIZE}(\psi_2) + 1$ and $\deg(\phi) = \max\{\deg(\psi_1), \deg(\psi_2)\}$.*
4. *$\phi = \psi_1 \cdot \psi_2$, where both of ψ_1, ψ_2 are arithmetic formulas over $x_1, \ldots, x_n$, in which case $\mathcal{SIZE}(\phi) = \mathcal{SIZE}(\psi_1) + \mathcal{SIZE}(\psi_2) + 1$ and $\deg(\phi) = \deg(\psi_1) + \deg(\psi_2)$.*

We remark that in Definition 7.1 we did not specify an underlying field $\mathbb{F}$. A formula ϕ can be evaluated as a polynomial over any field $\mathbb{F}$ of our choice.

Representation of formulas. We now fix a canonical way of presenting formulas as Boolean strings. Note that a formula ϕ from Definition 7.1 can be organized as a tree, such that each node v is either (1) A leaf that corresponds to a variable x_i or to a constant $\sigma \in \{0,1\}$; or (2) An intermediate node corresponding to a sub-formula $-\psi_0$, and has one child w such that w corresponds to the sub-formula ψ_0; or (3) An intermediate node corresponding to a sub-formula $\psi_1 \ \mathsf{op} \ \psi_2$, where $\mathsf{op} \in \{+, \cdot\}$, and has two children w_1 and w_2 such that w_i corresponds to the sub-formula ψ_i for each $i \in [2]$. In the second case, we say the node has operator type $-$, and the third case, we say the node has operator type op.

In a formula ϕ with m nodes, we will represent each node by an unique identifier $i \in [m]$, and the node 1 is the root. Then ϕ can be described as a list of m descriptions, one for each node: In the description for node v, we first specify whether its a leaf or an intermediate node. If it is a leaf, we then specify its corresponding variable or constant; if it is an intermediate node, we specify its operator and indices of children.

Families of arithmetic formulas of low degree. A formula is a syntactic model over a fixed number of variables, and we will be interested in formula families evaluated over a sequence of finite fields (i.e., one formula and one field for each input length). Throughout the section, we denote formula families by $F = \{F_n\}_{n\in\mathbb{N}}$ and use the convention that F_n is over n variables. We also consider a "post-processing" operation, which maps the output of the formula to a Boolean string. In more detail:

Definition 7.2 (defining triplets)**.** *Let $p\colon \mathbb{N} \to \mathbb{N}$ be a function mapping integers to prime powers, and let $g = \{g_n\}$ be such that $g_n\colon \mathbb{F}_{p(n)} \to \{0,1\}^*$. Let $F = \{F_n\}$ be a family of arithmetic formulas. We define $\Pi = \Pi^{F,p,g} = \{\Pi_n\}$ such that Π_n maps $x \in \mathbb{F}_{p(n)}^n$ to $g_n(F_n(x))$. We say (F,p,g) is a Π-defining triplet with degree $d(n)$ and size $s(n)$ if:*

1. *(**Arithmetic setting.**) The field size satisfies $d(n)^2 \leq p(n) \leq \mathrm{poly}(n)$, and either $p(n)$ is a prime for all $n \in \mathbb{N}$, or there exists a constant prime p_0 such that $p(n)$ is a power of p_0 for all $n \in \mathbb{N}$. Also, there is a logspace-uniform family of circuits of polynomial size and $\mathrm{polylog}(n)$ depth that maps 1^n to the integer $p(n)$.*
2. *(**Formula family.**) For every $n \in \mathbb{N}$ it holds that F_n has degree at most $d(n)$ and size at most $s(n)$, and the family F is logspace-uniform.*
3. *(**Post-processing.**) The function $g_n\colon \mathbb{F}_{p(n)} \to \{0,1\}^*$ is computable in space $O(\log n)$, given a representation of $\mathbb{F}_{p(n)}$.*

7.1 Worst-case to average-case reduction for arithmetic formulas

The main lemma that we will need for the proof of Theorem 1.5, which we now state, asserts that there is a worst-case to mild average-case reduction for any problem $\Pi = \Pi^{F,p,g}$ defined by low-degree arithmetic formulas. We first state and prove this lemma, and then deduce Theorem 1.5 from this lemma, relying on Theorem 6.1.

Lemma 7.3 (worst-case to mild average-case reduction for low-degree arithmetic formulas)**.** *There is a universal constant $\gamma \in \mathbb{N}$ such that the following holds. Let $c,k \in \mathbb{N}$ be constants such that c is sufficiently large. Suppose there is a Π-defining triplet (F,p,g) with degree n^k and size $\mathrm{poly}(n)$ such that $\Pi = \Pi^{F,p,g} \notin \mathtt{i.o.}\mathcal{BPTIME}[n^c]$. Then, there exists a function $\bar{F}\colon \{0,1\}^* \to \{0,1\}^*$ such that*

$$\bar{F} \in \mathsf{lu}\text{-}\mathcal{CKT}[\mathrm{poly}(n), \mathrm{polylog}(n)] \setminus \mathtt{i.o.}\text{-}\mathsf{avg}_{(1-n^{-2k})}\text{-}\mathcal{BPTIME}[n^{c-\gamma\cdot k}]\,,$$

and $\bar{F}$ has an ℓ-strongly $(1/10)$-tolerant instance checker running in time n^b, for some $\ell(n) = \Omega(n)$ and a universal constant b.

Proof. Our first step is to argue that a representation of the field $\mathbb{F} = \mathbb{F}_{p(n)}$ can be constructed efficiently. Specifically, let $\pi_n^{\mathbb{F}}$ be a Boolean string that describes $\mathbb{F} = \mathbb{F}_{p(n)}$, which consists of the prime p_0 such that $p(n) = (p_0)^m$ (it is possible that $p_0 = p(n)$ and $m = 1$) and an irreducible polynomial over $\mathbb{F}_{p_0}$ of degree m. The string $\pi_n^{\mathbb{F}}$ is of length $O(\log n)$, and we pad $\pi_n^{\mathbb{F}}$ to be of length 2^{ℓ_π} where $\ell_\pi = \log\log(n) + O(1)$.

Claim 7.3.1. *There is a logspace-uniform circuit family of polynomial size and $\mathrm{polylog}(n)$ depth that gets input 1^n and prints $\pi_n^{\mathbb{F}}$.*

Proof. By our assumption, a logspace-uniform circuit family of polynomial size and depth $\mathrm{polylog}(n)$ can print $p(n)$. Recall that either $p(n)$ is a prime for all $n \in \mathbb{N}$, or $p(n)$ is a power of a fixed constant prime p_0 for all $n \in \mathbb{N}$. In the first case, we are

done. In the second case, we need to find an irreducible polynomial of degree m; the algorithm of Shoup [Sho90] runs in time $\text{poly}(p_0, \log n) = \text{polylog}(n)$, and can thus be (naively) implemented by logspace-uniform circuits of size and depth $\text{polylog}(n)$. □

Next, consider the polynomial $P_n \colon \mathbb{F}^n \to \mathbb{F}$ that is obtained by evaluating the formula F_n over the field $\mathbb{F}$ in the obvious way, and note that its degree is at most n^k. We argue that the polynomial family $\{P_n\}$ can be computed by logspace-uniform circuits of polynomial size and depth $\text{polylog}(n)$:

Lemma 7.3.2. *The polynomial family $\{P_n\}_{n\in\mathbb{N}}$ can be computed by logspace-uniform circuits of $\text{polylog}(n)$ depth and polynomial size.*

The proof of Lemma 7.3.2 amounts to balancing the formula F_n by uniform circuits of polylogarithmic depth and polynomial size, in order to evaluate P_n by such circuits. We defer the proof to Section 7.1.1.

An auxiliary hard problem. To construct our hard problem we convert P_n into a Boolean function whose truth table contains $\pi_n^{\mathbb{F}}$ as a substring; this will allow our worst-case to average-case reduction to quickly obtain a description of $\mathbb{F}$.

Specifically, for $\ell_n = \left\lceil \log^2(n) \right\rceil$, we define the following mapping $\pi_n \colon \{0,1\}^{\ell_n} \to \mathbb{F}$: Given $x \in \{0,1\}^{\ell_n}$, interpret it as an integer $x \in [2^{\ell_n}]$, let $i = x \bmod p(n)$, and output the i^{th} element in $\mathbb{F}$. [41] For $m = n \cdot \ell_n + 1$ and $\ell_{\mathbb{F}} = \lceil \log(|\mathbb{F}|) \rceil$, we define a function $H_n \colon \{0,1\}^m \times [\ell_{\mathbb{F}}] \mapsto \{0,1\}$, as follows:

1. Given an input $(\lambda, x, i) \in \{0,1\} \times \{0,1\}^{n\cdot\ell_n} \times [\ell_{\mathbb{F}}]$, partition x into n consecutive blocks $x^{(1)}, \ldots, x^{(n)} \in \{0,1\}^{\ell_n}$.
2. If $\lambda = 0$, output the i^{th} bit in the binary representation of $P_n(\pi_n(x^{(1)}), \ldots, \pi_n(x^{(n)}))$.
3. If $\lambda = 1$, let j be the integer represented by the first ℓ_π bits of x and output $(\pi_n^{\mathbb{F}})_j$.

Let $H \subseteq \{0,1\}^*$ be the language defined by the H_n's (i.e., H is non-trivially defined only on input lengths of the form $n = m + \lceil \log(\ell_{\mathbb{F}}) \rceil$) We argue that H is hard, relying on the hardness of Π. The idea is that if a fast algorithm can compute H_n correctly on a $1 - n^{-(k+1)}$ fraction of the inputs, then we can use it to quickly compute a representation of $\mathbb{F}$, then apply a worst-case to average-case reduction for the low-degree polynomial P_n (i.e., Theorem 3.6) to inputs for H_n of the form $(1, x, i)$, and finally compute g_n quickly.

Lemma 7.3.3. *There exists a universal constant $c' > 1$ such that for every probabilistic algorithm running in time $n^{c-c'\cdot k}$ and every sufficiently large $n \in \mathbb{N}$, for at least an $n^{-(k+1)}$ fraction of inputs $z \in \{0,1\}^m \times [\ell_{\mathbb{F}}]$ it holds that $\Pr[A(z) = H(z)] < 2/3$.*

We defer the full proof of Lemma 7.3.3 to Section 7.1.2. Observe that

$$H \in \text{lu-}\mathcal{CKT}[\text{poly}(n), \text{polylog}(n)] \,,$$

relying on Claim 7.3.1 and Lemma 7.3.2.

[41] We assume that elements in $\mathbb{F}$ are ordered lexicographically.

A problem that is hard on all input lengths. The only missing piece is that H is defined only on some input lengths, whereas we are interested in a function that is hard on all input lengths. We thus reduce H to the corresponding function F from the "furthermore" part of Proposition 4.4, which is also computable in $\mathsf{lu}\text{-}\mathcal{CKT}[\mathrm{poly}(n), \mathrm{polylog}(n)]$. We will rely on the following lemma:

Lemma 7.4 (preserving mild average-case hardness on all input lengths under reductions using a strongly tolerant instance checker)**.** *Let $H \subseteq \{0,1\}^*$ and $\ell(n) \leq \mathrm{polylog}(n)$ such that for every probabilistic algorithm A that runs in time $n^{c-c'\cdot k}$ on inputs of length $n \cdot \ell(n)$ and every sufficiently large $n \in \mathbb{N}$, the probability over $x \in \{0,1\}^{n\cdot\ell(n)}$ that $\Pr[A(x) = L^{(0)}(x)] \geq 2/3$ is less than $1 - n^{-(k+1)}$. Let $F\colon \{0,1\}^* \to \{0,1\}^*$ such that for two constants $a, b > 1$:*

1. *Deciding H is reducible in linear time to computing F via an input-extending reduction that maps inputs of length n to inputs of length $a \cdot n$.*
2. *The function F has an (n/a)-strongly $(1/10)$-tolerant same-length instance checker with running time n^b.*

Then, the natural S-extension of F is not in $\mathtt{i.o.}\text{-}\mathsf{avg}_{(1-n^{-(k+1)}/100)}\text{-}\mathcal{BPTIME}[n^{c-a\cdot k-b-2}]$, where $S = \{a \cdot n_0 : n_0 \in \mathbb{N}\}$.

We defer the proof of the lemma above to Appendix A (see the "furthermore" part of Lemma A.2). Now, recall that H is reducible to F via an input-extending reduction mapping inputs of length n to inputs of length $a \cdot n$, and that F has a (n/a)-strongly $(1/10)$-tolerant same length instance checker running in time n^b, for universal constants $a, b > 1$. Thus, denoting the S-extension of F by $\bar{F}$, we have that

$$\bar{F} \in \mathsf{lu}\text{-}\mathcal{CKT}[\mathrm{poly}(n), \mathrm{polylog}(n)] \setminus \mathtt{i.o.}\text{-}\mathsf{avg}_{(1-n^{-(k+2)})}\text{-}\mathcal{BPTIME}[n^{c-\gamma\cdot k}]\,,$$

for a universal constant $\gamma > 1$. ■

7.1.1 Proof of Lemma 7.3.2

By Claim 7.3.1, we can assume that we have a representation of $\mathbb{F}$. We will follow the standard algorithm that balances a polynomial-size arithmetic formula into depth $O(\log n)$ (see, e.g., [SY10, Theorem 2.6]), and explain how it can be implemented by logspace-uniform circuits of polylogarithmic depth and polynomial size.

Recall that the partial derivative of a formal polynomial $f(x_1, ..., x_n)$ with respect to variable x_i, denoted $\partial_{x_i}(f)$ is a polynomial defined recursively as follows: When f does not contain x_i we have $\partial_{x_i}(f) = 0$; when f is the monomial x_i we have $\partial_{x_i}(f) = 1$; and we have $\partial_{x_i}(f+g) = \partial_{x_i}(f) + \partial_{x_i}(g)$ and $\partial_{x_i}(f \cdot g) = \partial_{x_i}(f) \cdot g + f \cdot \partial_{x_i}(g)$.

For a formula Φ and node w, we denote by Φ_w the subformula rooted at w; and for a field element y, we denote by $\Phi\restriction_{w=y}$ the formula obtained by fixing Φ_w in Φ to be the constant y. When we write $\partial_w(\Phi)$, we think of the formula Φ' over variables $x_1, ..., x_n$ and y that outputs $\Phi\restriction_{w=y}(x_1, ..., x_n)$ and derive it in y (i.e., $\partial_w(\Phi) = \partial_y(\Phi')$. Consider the following procedure $\mathsf{Simplify}$, which we will use as a recursive step:

1. **Input:** A description of a size-s formula Φ over $x_1, ..., x_n$ as a bit-string.
2. **Execution steps:**
   (a) Find a node w in Φ such that $\mathcal{SIZE}(\Phi_w) \in [s/3, 2s/3]$.

(b) Compute a description of Φ_w, of $\Phi\restriction_{w=0}$, and of $\partial_w(\Phi)$.

3. **Output:** The expression $\partial_w(\Phi) \cdot \Phi_w + \Phi\restriction_{w=0}$.

A standard analysis (again see, e.g., [SY10, Theorem 2.6]) shows that $\Phi = \partial_w(\Phi) \cdot \Phi_w + \Phi\restriction_{w=0}$, and thus the output expression computes the same function as the input formula; and that each of the three formulas $\partial_w(\Phi)$ and Φ_w and $\Phi\restriction_{w=0}$ is of size at most $2s/3$. We argue that Simplify can be performed in parallel:

Claim 7.4.1. *There is a logspace-uniform family of circuits of size* $\mathrm{poly}(n,s)$ *and depth* $\mathrm{polylog}(n,s)$ *that computes* Simplify.

Proof. As a first step we write down the adjacency matrix of the directed graph induced by Φ, where the direction points from parent to children, and iteratively square this matrix for $O(\log(n \cdot s))$ times. This can be done by logspace-uniform circuits of depth $O(\log(n \cdot s)^2)$ and size $\mathrm{poly}(n,s)$. (Recall that in the description of Φ, the description of each internal node contains the indices of the two children.) The resulting matrix allows us to quickly decide, for each pair (w,v), whether $v \in \Phi_w$.

Now, for Item (2a), we enumerate over all nodes in parallel and count the number of descendants of each node (by enumerating again over all nodes in parallel and using the matrix above), then take the lexicographically first node w (i.e., the node w whose index is smallest) with between $s/3$ and $2s/3$ descendants.

As for Item (2b), to compute a description of Φ_w (resp., of $\Phi\restriction_{w=0}$) from the description of Φ we just need to eliminate all the nodes that are not children of w (resp., that are children of w), which we do again by enumerating all nodes in parallel and checking each node. To compute a description $\partial_w(\Phi)$, note that the following iterative procedure (which we do not execute as-is) would yield this description:

1. Iterate from w upwards in Φ, maintaining a formula that is initiated to $f_0 = 1$.
2. In a given iteration i, let f_{i-1} be the formula after the previous iteration, let u_i be the parent node encountered when traversing up, let $v_{i,1}$ be the node from which we arrived, and let $v_{i,2}$ be the sibling of $v_{i,1}$ if u_i has two children.
   - If u_i is labeled with $+$, let $f_i = f_{i-1}$.
   - If u_i is labeled with $-$, let $f_i = -f_{i-1}$.
   - If u_i is labeled with $\times$, let $f_i = f_{i-1} \cdot \Phi_{v_{i,2}}$.
3. After the final iteration $i \leq s$ (such that u_i is the root) we output f_i.

To see that this procedure yields $\partial_w(\Phi)$, apply the recursive definition of ∂ to u_i at each iteration, and note that we always have $\partial_w(v_{i,2}) = 0$ (because Φ is a formula).

The point is that to compute $\partial_w(\Phi)$ we do not need to iteratively execute the procedure above. Specifically, unravelling the recursion in this procedure, we have that

$$\partial_w(\Phi) = (-1)^{|\{u_i \in \mathcal{S}_w : \mathsf{Label}(u_i) = -\}|} \cdot \prod_{u_i \in \mathcal{S}_w : \mathsf{Label}(u_i) = \times} v_{i,2} \,, \tag{7.1}$$

where $\mathcal{S}_w$ is the set of ancestors of w and $\mathsf{Label}(u)$ is the operator labelling the node u. Thus, to compute $\partial_w(\Phi)$ we can enumerate over all nodes u in parallel, check whether $w \in \Phi_u$, and compute the expression in Eq. (7.1) (by either multiplying or increasing the counter for the power of -1), all by logspace-uniform circuits of depth $\mathrm{polylog}(n,s)$ and size $\mathrm{poly}(n,s)$. □

Now, our algorithm maintains an arithmetic expression f that adds and multiplies formulas such that f computes Φ. In the beginning we have that $f = \Phi$, and then the algorithm iteratively calls Simplify, for $t = \Theta(\log(s))/\text{loglog}(s))$ times, in each iteration applying Simplify to each of the elements in the current expression f.

Recall that each application of Simplify to a formula of size s' yields an expression with three formulas, each of them of size at most $2s'/3$. Thus, after t steps the final expression contains $\text{poly}(n)$ formulas, each of them of size $\text{polylog}(s)$. The algorithm evaluates each of the formulas, and then computes the arithmetic expression of the outcomes. This procedure can be done by logspace-uniform circuits of depth $\text{polylog}(n, s)$ and size $\text{poly}(n, s)$.

7.1.2 Proof of Lemma 7.3.3

For convenience, we denote the domain of H_n by $\mathbb{D}_n = \{0,1\}^m \times [\ell_{\mathbb{K}}]$, and denote $\mathbb{D}_n^0 = \{0\} \times \{0,1\}^{n \cdot \ell_n} \times [\ell_{\mathbb{K}}]$ and $\mathbb{D}_n^1 = \{1\} \times \{0,1\}^{n \cdot \ell_n} \times [\ell_{\mathbb{K}}]$.

Assume towards a contradiction that there is a probabilistic algorithm A running in time T_A such that for infinitely many $n \in \mathbb{N}$, for more than $1 - \epsilon$ fraction of inputs $z \in \mathbb{D}_n$ it holds that $\Pr[A(z) = H_n(z)] \geq 2/3$, where $\epsilon = n^{-(k+1)}$. By naive error reduction, we can assume that for every such z it holds that $\Pr[A(z) = H_n(z)] \geq 1 - 2^{-n}$, and we denote the set of z's for which this holds by $Z \subseteq \mathbb{D}_n$.

We construct an algorithm B for Π_n as follows. The input to the problem is $x \in \mathbb{F}_{p(n)}^n$, represented as a binary string. The algorithm B first computes a description of $\mathbb{F} = \mathbb{F}_{p(n)}$ by using A to obtain the string $\pi_n^{\mathbb{F}}$. Specifically, it randomly chooses $j \in \{0,1\}^{n \cdot \ell_n - \ell_\pi}$ and $u \in [\ell_{\mathbb{F}}]$ and queries A on the 2^{ℓ_π} inputs $\{(1, i \circ j, u)\}_{i \in [2^{\ell_\pi}]}$. By our assumption it holds that $\Pr_{z \in \mathbb{D}_n^1}[z \in Z] \geq 1 - 2\epsilon$, and therefore

$$\Pr_{j,u}\left[\exists i : (1, i \circ j, u) \notin Z\right] \leq 2^{\ell_\pi} \cdot (2\epsilon) \leq 1/n \ ,$$

relying on the fact that $2^{\ell_\pi} = O(\log(n))$ and on our choice of ϵ. Thus, with high probability at this point B obtained the string $\pi_n^{\mathbb{F}}$ and thus a description of $\mathbb{F}$.

At this point B runs the decoder RM-Dec from Theorem 3.6 with input x and degree parameter $d = n^k$, while answering its queries as follows. Given query $q = (q_1, ..., q_n) \in \mathbb{F}^n$,

1. For each $i \in [n]$, let $j \in [p(n)]$ such that q_i is the j^{th} element in $\mathbb{F}$. Choose a random shift $r_i \in \left\{0, 1, ..., \left\lfloor \frac{2^{\ell_n} - p(n)}{p(n)} \right\rfloor \right\}$, and let $\tilde{q}_i = r_i \cdot p(n) + q_i \in [2^{\ell_n}]$.
2. For each $u \in [\ell_{\mathbb{F}}]$, run A on input $(0, (\tilde{q}_1, ..., \tilde{q}_n), u)$, and return the element represented by the $\ell_{\mathbb{F}}$ bits.

Now, recall that RM-Dec makes $d + 1$ non-adaptive queries, and that each query q is (marginally) uniformly distributed in $\mathbb{F}^n$. Therefore, each $\tilde{q}_i$ is uniformly distributed in a set of density at least $(1 - p(n)/2^{\ell_n})$ in $[2^{\ell_n}]$,[42] and hence each query $\tilde{q}$ of RM-Dec is uniformly distributed in a set of density $(1 - p(n)/2^{\ell_n})^n \geq 1 - n^{-\log(n)/2}$. Thus, the algorithm A answers each query $(0, \tilde{q}, k)$ correctly, with probabilty at least $1 - 3\epsilon$, where the probability is over the random coins of RM-Dec, of the random choice of shifts r_i's for each query, and of the random coins of A.

[42] If $p(n) | 2^{\ell_n}$ then $\tilde{q}_i$ is uniformly distributed in $[2^{\ell_n}]$, but otherwise $\tilde{q}_i$ is uniformly distributed in a set of size at least $p(n) \cdot \left(\frac{2^{\ell_n} - p(n)}{p(n)} - 1\right) + p(n) = 2^{\ell_n} - p(n)$.

It follows that with probability at least $1 - 3\epsilon \cdot (d+1) \cdot \ell_{\mathbb{F}} > 1 - 1/100$ all queries of RM-Dec are answered correctly, where the inequality relied on our choice of ϵ. Hence, with probability at least $9/10$ over choice of random coins for A and of shifts vector $\bar{r} \in (\left\{0, ..., \left\lfloor 2^{\ell_n - p(n)} p(n) \right\rfloor\right\})^{n \cdot (d+1)}$, the probability over coins of RM-Dec that all of its queries are answered correctly is at least $9/10$. In this case, with high probability RM-Dec outputs $P(x)$. Finally, B outputs $g_n(P(x))$. The running time of B is at most

$$\tilde{O}\left(2^{\ell_\pi} \cdot T_A + \mathrm{poly}(d, n, \log(|\mathbb{F}|)) \cdot T_A \cdot \log(|\mathbb{F}|) + d\right) < T_A \cdot n^{c' \cdot k} ,$$

where $c' > 1$ is a universal constant. This contradicts the hypothesized hardness of Π (for time n^c) if $T_A < n^{c - c' \cdot k}$.

7.2 Proof of Theorem 1.5

We now state and prove Theorem 1.5. The following statement is more refined than the original one, since it allows any fixed polynomial degree (rather than only n^2), and refers to derandomization of $\mathcal{RTIME}[n^b]$ for any specific constant $b > 1$, relying on hardness for probabilistic time n^c where c is sufficiently large. (The original statement follows since it assumes hardness for all values of $c \in \mathbb{N}$.)

Theorem 7.5 (derandomization from worst-case fine-grained hardness for low-degree arithmetic formulas)**.** *For every $a, b, k \in \mathbb{N}$, there exists a constant $c \geq 1$ such that the following holds. Suppose there is a Π-defining triple (F, p, g) with degree n^k and size $\mathrm{poly}(n)$ such that $\Pi = \Pi^{F,p,g} \notin \mathtt{i.o.}\mathcal{BPTIME}[n^c]$. Then, $\mathcal{RTIME}[n^b] \subseteq \mathsf{avg}_{(1-n^{-a})}\text{-}\mathcal{P}$.*

Proof. Let $c \geq 1$ be a sufficiently large constant to be determined later. By Lemma 7.3, there exists

$$\bar{F} \in \mathsf{lu}\text{-}\mathcal{CKT}[\mathrm{poly}(n), \mathrm{polylog}(n)] \setminus \mathtt{i.o.}\text{-}\mathsf{avg}_{(1-n^{-2k})}\text{-}\mathcal{BPTIME}[n^{c-\gamma \cdot k}] ,$$

such that $\bar{F}$ has an ℓ-strongly $(1/10)$-tolerant instance checker running in time $n^{c'}$, for some $\ell(n) = \Omega(n)$ and a universal constant c'. Using Proposition 6.4 with parameters $c_1 = 1$ and $c_2 = 2k$ and $c_3 = c'$ and a and b, we deduce that $\mathcal{RTIME}[n^b] \subseteq \mathsf{avg}_{1-n^{-a}}\text{-}\mathcal{P}$, assuming that $c - \gamma \cdot k$ is sufficiently large (i.e., larger than $f(a, b, c_1, c_2, c_3) = f(a, b, c_2)$ for some universal function f). We ensure the latter condition by choosing $c = c(a, b, k)$ to be sufficiently large. ∎

To see how the general hardness hypotheses in Theorem 7.5 can be instantiated for specific natural functions, let us state a corresponding corollary for k-CLIQUE, or k-OV, or k-SUM (for definitions see, e.g., [Wil18]).

Corollary 7.6 (derandomization from hardness of k-clique and k-OV and k-SUM)**.** *Assume that for every constant $c \in \mathbb{N}$ there exists $k \in \mathbb{N}$ such that counting k-cliques, or counting k-OV, or counting k-SUM cannot be done in probabilistic time n^c, even infinitely often. Then, $\mathcal{RP} \subseteq \mathsf{avg}_{(1-1/n^\kappa)}\text{-}\mathcal{P}$ for every constant $\kappa \in \mathbb{N}$.*

Acknowledgements

Lijie Chen is supported by NSF CCF-2127597 and an IBM Fellowship. Ron Rothblum was funded by the European Union. Views and opinions expressed are however those

of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Part of this work was conducted while Roei Tell was supported by the National Science Foundation under grant number CCF-1445755 and under grant number CCF-1900460.

References

[ABK+06] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. "Power from random strings". In: *SIAM Journal of Computing* 35.6 (2006), pp. 1467–1493.

[BRS+17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. "Average-case fine-grained hardness". In: *Proc. 49th Annual ACM Symposium on Theory of Computing (STOC)*. 2017, pp. 483–496.

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

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

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

[CT21] 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.

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

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

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

[GS92] Peter Gemmell and Madhu Sudan. "Highly Resilient Correctors for Polynomials". In: *Inf. Process. Lett.* 43.4 (1992), pp. 169–174.

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

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

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

[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.

[IKV18] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. "The power of natural properties as oracles". In: *Proc. 33rd Annual IEEE Conference on Computational Complexity (CCC)*. 2018, Art. No. 7, 20.

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

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

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

[KKO13] Adam Klivans, Pravesh Kothari, and Igor Oliveira. "Constructing Hard Functions Using Learning Algorithms". In: *Proc. 28th Annual IEEE Conference on Computational Complexity (CCC)*. 2013, pp. 86–97.

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

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

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

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

[OS17] Igor C. Oliveira and Rahul Santhanam. "Pseudodeterministic constructions in subexponential time". In: *Proc. 49th Annual ACM Symposium on Theory of Computing (STOC)*. 2017, pp. 665–677.

[San09] Rahul Santhanam. "Circuit lower bounds for Merlin-Arthur classes". In: *SIAM Journal of Computing* 39.3 (2009), pp. 1038–1061.

[Sha10] Ronen Shaltiel. "Typically-Correct Derandomization". In: *SIGACT News* 41.2 (2010), 57–72.

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

[Sho90] Victor Shoup. "New algorithms for finding irreducible polynomials over finite fields". In: *Mathematics of Computation* 54.189 (1990), pp. 435–447.

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

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

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

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. Vol. 1001. Lecture Notes in Computer Science. Springer, 1995. ISBN: 3-540-60615-7.

[SY10] Amir Shpilka and Amir Yehudayoff. "Arithmetic Circuits: A survey of recent results and open questions". In: *Found. Trends Theor. Comput. Sci.* 5.3-4 (2010), pp. 207–388.

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

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

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

[VSB+83] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. "Fast parallel computation of polynomials using few processors". In: *SIAM Journal of Computing* 12.4 (1983), pp. 641–644.

[Wil18] Virginia Vassilevska Williams. "On some fine-grained questions in algorithms and complexity". In: *Proc. of the International Congress of Mathematicians—Rio de Janeiro 2018. Vol. IV. Invited lectures*. 2018, pp. 3447–3487.

A Preserving strong hardness using tolerant instance checkers

In this appendix we prove several technical results asserting that if a function f is hard, and is reducible to another function $\bar{f}$, then $\bar{f}$ is also hard. Of course, such a statement is trivial, but our goal is to preserve strong notions of hardness, such as hardness on *almost all input lengths* and hardness *on average-case inputs*. Moreover, we will sometimes define $\bar{f}$ using error-correcting codes, and claim stronger average-case hardness for $\bar{f}$, compared to f. We prove claims of this form for several different settings, and in all settings we rely on the existence of instance checkers or of tolerant instance checkers either for f or for $\bar{f}$.

The proofs in this appendix follow standard technical ideas. However, in some cases we will apply these ideas to new notions, such as strongly tolerant instance checkers, and in all cases fleshing out the technical details is a bit tedious.

A.1 Reducing to an instance-checkable problem

In our first setting we start from a hard problem $L^{(0)}$, reduce it to an instance-checkable problem L, and argue that L is hard on almost all input lengths. That is:

Lemma A.1 (preserving hardness on all input lengths under reductions using an instance checker)**.** *Fix time-computable increasing functions $T, t\colon \mathbb{N} \to \mathbb{N}$. Let $L^{(0)} \notin$*

$\mathtt{i.o.}\mathcal{BPTIME}[T]$, *and let* L *such that* $L^{(0)}$ *is reducible in linear time to* L *via a reduction that maps inputs of length* n *to inputs of length* $c \cdot n$*, where* $c > 1$ *is a constant. Further assume that* L *has a same-length instance-checker whose running time is* t*. Then, for* $S = \{c \cdot n_0 : n_0 \in \mathbb{N}\}$*, the natural* S*-extension of* L *is not in* $\mathtt{i.o.}\mathcal{BPTIME}[T']$*, for* $T'(n) = T(n/2c)/\tilde{O}(t(n))$*.*

Proof. Let $\bar{L}$ be the natural S-extension of L. Assuming towards a contradiction that $\bar{L} \in \mathtt{i.o.}\mathcal{BPTIME}[T']$, we show that $L^{(0)} \in \mathtt{i.o.}\mathcal{BPTIME}[T]$. Let M' be a probabilistic T'-time machine and let $S' \subseteq \mathbb{N}$ be an infinite set of input lengths on which M' solves $\bar{L}$. For every $n \in S$, denote $I_n = [n, n+c)$, and observe that for every $n' \in S'$ there exists $n \in S$ such that $n' \in I_n$. Therefore, there are infinitely many $n \in S$ such that $I_n \cap S' \neq \emptyset$.

We define M that solves $L^{(0)}$ infinitely often as follows. The machine gets input $x_0 \in \{0,1\}^{n_0}$, and we assume that n_0 is such that $I_n \cap S' \neq \emptyset$, where $n = c \cdot n_0$ (we do not attempt to prove correctness of M on inputs not of such lengths). The machine applies the linear-time reduction of deciding $L^{(0)}$ at x_0 to deciding L at $x \in \{0,1\}^n$, and for $i = 0, ..., c-1$ it runs the instance checker on input x. In iteration i, it answers each query $q_j \in \{0,1\}^n$ of the instance checker by running M' on input $q_j 0^i$. Moreover, whenever the machine simulates each of the two algorithms (i.e., M' and the instance checker), it repeat the algorithm for $O(\log(t))$ times to reduce the error to $1/O(t(n))$.[43] It returns an answer $\sigma \in \{0,1\}$ if the answers of the instance checker on all inputs are in the set $\{\sigma, \perp\}$, and otherwise we return $\perp$.

With high probability, for every $i \in \{0, ..., c-1\}$ it holds that the answer of the instance checker is in $\{L(x), \perp\}$; and for i such that $n + i \in (I_n \cap S')$, all queries of the instance checker to M' were answered by L, in which case the instance checker outputs $L(x) = L^{(0)}(x_0)$. The running time of the algorithm for $L^{(0)}$ on input length n_0 is

$$\mathrm{polylog}(t(n)) \cdot (t(n+c) \cdot T'(n+c) \leq \tilde{O}(t(2c \cdot n_0)) \cdot T'(2c \cdot n_0) \ ,$$

which is at most $T(n_0)$, yielding a contradiction. ■

The foregoing claim referred to worst-case hardness, and we now prove a similar claim that refers to mild average-case hardness. That is, we assume that every efficient algorithm for $L^{(0)}$ errs on at least (say) 1% of the inputs, and deduce that efficient every algorithm for L also errs on 1% of inputs. To obtain this conclusion we rely on the stronger assumption that L has a *tolerant* instance checker, and in fact a *strongly tolerant* instance checker (see Definitions 3.12 and 3.13).

For our application in Section 6 it will be more convenient to prove this claim referring to the particular parameter setting of polynomial-time algorithms and polynomial-time reductions, and replace languages with functions that have multiple output bits.

Lemma A.2 (preserving mild average-case hardness on all input lengths under reductions using a strongly tolerant instance checker)**.** *For every three constants* $c_1, c_2, c_3 \geq 1$ *and non-increasing* $\delta, \mu : \mathbb{N} \to (0,1)$ *the following holds. Let* $L^{(0)} \notin \mathtt{i.o.}\text{-}\mathsf{avg}_{(1-\delta)}\text{-}\mathcal{BPTIME}[n^{c_1}]$*, and let* $f : \{0,1\}^* \to \{0,1\}^*$ *such that:*

1. *Deciding* $L^{(0)}$ *is reducible in linear time to computing* f *via an input-extending reduction that maps inputs of length* n *to inputs of length* $c_3 \cdot n$*.*

[43] To be more accurate, we run the instance checker for $O(\log t)$ times and output a value $\sigma \in \{0,1\}$ only if all runs yielded the value σ; and we run M' for $O(\log t)$ times and output the majority value.

2. *The function f has an ℓ-strongly μ-tolerant same-length instance checker with running time n^{c_2}, where $\ell(n) = n/c_3$.*

Then, for $S = \{c_3 \cdot n_0 : n_0 \in \mathbb{N}\}$, the natural S-extension of f is not in $\texttt{i.o.-avg}_{(1-\delta')}$-$\mathcal{BPTIME}[n^c]$, where $\delta' = (\mu/10) \cdot \delta$ and $c = c_1 - c_2 - 1$.

Furthermore, the conclusion holds with $c = c_1 - c_2 - 2$ if we replace the hypothesis "$L^{(0)} \notin \texttt{i.o.-avg}_{(1-\delta)}$-$\mathcal{BPTIME}[n^{c_1}]$" by the following assumption: For some $\ell(n) \leq \text{polylog}(n)$ such that $\ell(n+1) \leq \ell(n) + 1$, for every probabilistic algorithm A that runs in time n^{c_1} on inputs of length $n \cdot \ell(n)$ and every sufficiently large $n \in \mathbb{N}$, the probability over $x \in \{0,1\}^{n \cdot \ell(n)}$ that $\Pr[A(x) = L^{(0)}(x)] \geq 2/3$ is less than $1 - \delta(n)$.

Proof. We first prove the main part of the claim and then explain how to extend the proof to demonstrate the "furthermore" part. Let $\bar{f}$ be the natural S-extension of f, let M' be a probabilistic n^c-time machine and let $S' \subseteq \mathbb{N}$ be an infinite set such that for every $n \in S'$ it holds that M' computes $\bar{f}$ on $1 - \delta'(n)$ of the n-bit inputs. Also, for every $n \in S$ denote $I_n = [n, n + c_3]$. We will use the fact that every integer (in particular, every integer in S') is contained in I_n for some $n \in S$ (since $c_3 \cdot (n_0 + 1) - c_3 \cdot n = c_3$).

We use M' to construct a machine M for L_0 as follows. For n_0 and $n = c_3 \cdot n_0$ such that $I_n \cap S' \neq \emptyset$, the machine M for $L^{(0)}$ gets input $x_0 \in \{0,1\}^{n_0}$, reduces the problem to computing f on $x \in \{0,1\}^n$, and for $i = 0, ..., c_3 - 1$ it runs the tolerant instance checker for f on input x, while answering its queries as follows. It first chooses uniformly at random a string $u_i \in \{0,1\}^i$; then, given any query $q_j \in \{0,1\}^n$, it simulates machine M' on input $q_j u_i$. (Note that the same string u_i is reused for all queries q_j in iteration i.) Again the machine uses naive error reduction for all algorithms, with $O(\log n)$ repetitions, to yield error $1/\text{poly}(n)$. If for some i the output of the instance checker was $\sigma \in \{0,1\}^*$ then M outputs the decision for x_0 that is obtained when $f(x) = \sigma$; otherwise, it outputs some default value (say, 0).

The running time of M is less than

$$O_{c_3}((n + c_3)^{c_2} \cdot (n + c_3)^c \cdot \text{polylog}(n)) < (n_0)^{c_2 + c + 1} = (n_0)^{c_1} ,$$

relying on our choice of $c = c_1 - c_2 - 1$. Also, with high probability all the outputs of the instance checker will be in the set $\{f(x), \perp\}$ (this is just as in the proof of Lemma A.1, relying on the soundness of the instance checker).

Now, let i such that $n' = n + i \in (I_n \cap S')$. On inputs of length $n' = n + i$, the machine M' computes $\bar{f}$ correctly on $1 - \delta'(n')$ of the inputs. For any fixed $x_0 \in \{0,1\}^{n_0}$, let

$$\begin{aligned} p(x_0) &= \Pr_{y \in \{0,1\}^{n-n_0}, u_i \in \{0,1\}^i} \left[M'(x_0 y u_i) \neq \bar{f}(x_0 y u_i) \right] \\ &= \Pr_{y \in \{0,1\}^{n-n_0}, u_i \in \{0,1\}^i} \left[M'(x_0 y u_i) \neq f(x_0 y) \right] \\ &= \Pr_{y \in \{0,1\}^{n-n_0}, u_i \in \{0,1\}^i} \left[\mathsf{Red}(M'(x_0 y u_i)) \neq L^{(0)}(x_0) \right] , \end{aligned}$$

where Red is the function implicit in the reduction of $L^{(0)}$ to f (i.e., $\mathsf{Red}(f(x_0 y)) = L^{(0)}(x_0)$).

Since $\mathbb{E}_{x_0 \in \{0,1\}^{n_0}} [p(x_0)] \leq \delta'(n')$, the probability over x_0 that $p(x_0) \geq 1/10\mu(n)$ is at most $(10/\mu) \cdot \delta'(n')$. For any x_0 in the set $X_0 = \{x_0 : p(x_0) < 1/10\mu(n)\}$, the probability over $u_i \in \{0,1\}^i$ that $\Pr_{y \in \{0,1\}^{n-n_0}} [M'(x_0 y u_i) \neq f(x_0 y)] > \mu(n)$ is at most $1/10$.

Thus, for any $x_0 \in X_0$, when machine M' gets input x_0, with probability at least $9/10$ over $u_i \in \{0,1\}^i$, the strongly tolerant instance checker gets access to a function $A(q) = M'(qu_i)$ that agrees with f on $1-\mu(n)$ of the inputs of the form $q = x_0 y \in \{0,1\}^{n_0} \times \{0,1\}^{n-n_0}$. Whenever this happens, with high probability the instance checker outputs $f(x_0 y)$, in which case $\mathsf{Red}(M'(x_0 y u_i)) = L^{(0)}(x_0)$. Thus, there is a set X_0 of density

$$1 - (10/\mu(n)) \cdot \delta'(n') = 1 - (10/\mu(n)) \cdot (\delta(n') \cdot \mu(n')/10) > 1 - \delta(n_0)$$

such that for every $x_0 \in X_0$, with high probability M' computes $L^{(0)}(x_0)$ in time n^{c_1}, a contradiction.

The "furthermore" part. We explain the necessary changes in the proof above. First, for every $n \in \mathbb{N}$ we define $I_n = \{n, n+1, ..., 2n\}$, and observe that:

Fact A.2.1. *For any sufficiently large $n' \in \mathbb{N}$ there exists $m \in \mathbb{N}$ such that $n' \in I_n$, where $n = c_3 \cdot m \cdot \ell(m)$.*

Proof. We argue that the interval-set $\{I_n : n = c_3 \cdot m \cdot \ell(m)\}_{m \in \mathbb{N}}$ covers all but finitely many integers. To see this, let $n = c_3 \cdot m \cdot \ell(m)$ and let $n' = c_3 \cdot (m+1) \cdot \ell(m+1)$, for a large enough $m \in \mathbb{N}$. Then,

$$\begin{aligned} n' &= c_3 \cdot (m+1) \cdot \ell(m+1) \\ &= c_3 \cdot m \cdot \ell(m+1) + c_3 \cdot \ell(m+1) \\ &< c_3 \cdot m \cdot (\ell(m)+1) + c_3 \cdot (\ell(m)+1) \\ &< 2c_3 \cdot m \cdot \ell(m)\ , \end{aligned}$$

which equals $2n$. □

It follows that if there is an infinite set $S' \subseteq \mathbb{N}$ such that for every $n \in S'$ it holds that M computes $\bar{f}$ on $1-\delta'(n)$ of the n-bit inputs, then there exist infinitely many $m \in \mathbb{N}$ such that $I_n \cap S' \neq \emptyset$, where $n = c_3 \cdot m \cdot \ell(m)$.

The construction of the machine M is essentially identical to the one above, except that it tries all integers $i \in \{0, ..., n-1\}$ rather than $i \in \{0, ..., c_3 - 1\}$. For every $n_0 = m \cdot \ell(m)$ such that $I_n \cap S' \neq \emptyset$ it succeeds in computing $L^{(0)}$ on more than $1 - \delta(n_0) > 1 - \delta(m)$ of the n_0-bit inputs, and its running time is at most

$$n \cdot (2n)^{c_2} \cdot (2n)^c \cdot \mathrm{polylog}(n) < \tilde{O}(n^{c_2+c+1}) < m^{c_1}\ ,$$

relying on the choice of $c = c_1 - c_2 - 2$. ■

A.2 Reducing from an instance-checkable problem to an encoded problem

In our second setting we start from a hard problem $L^{(1)}$ that is instance checkable, encode its truth-tables by a locally list-decodable code Enc to obtain a problem L, and argue that L is hard on average on almost all input lengths. In more detail:

Definition A.3 (encoded language)**.** *Let $L \subseteq \{0,1\}^*$ and let $\mathsf{Enc}\colon \{0,1\}^* \to \{0,1\}^*$ be a function such that for some polynomial p it holds that $|\mathsf{Enc}(f)| = 2^{p(\log(|f|))}$. We define the function $\mathsf{Enc}(L)$ as follows:*

1. *For every $n \in \mathbb{N}$ let f_n be the truth-table of L on n-bit inputs, and let $\bar{f}_n = \mathsf{Enc}(f_n)$. Then, the truth-table of $\mathsf{Enc}(L)$ on inputs of length $p(n)$ is $\bar{f}_n$.*

2. *For every $m \notin \{p(n) : n \in \mathbb{N}\}$ the function $\mathsf{Enc}(L)$ maps all m-bit strings to 0.*

Lemma A.4 (amplifying hardness on all input lengths using codes and an instance checker)**.** *Fix time-computable increasing functions $T, t, d \colon \mathbb{N} \to \mathbb{N}$ and $\rho \colon \mathbb{N} \to (0,1)$. Let $L^{(1)} \notin \mathtt{i.o.}\mathcal{BPTIME}[T]$ be a problem that is same-length instance-checkable in time t, let Enc be a systematic code that maps strings of length 2^n to strings of length $2^{c \cdot n}$ for some constant c, and that is locally list-decodable from agreement $\rho(n)$ in time d and with output-list size ℓ, where ℓ is an increasing function. Let $L = \mathsf{Enc}(L^{(1)})$ and $S = \{c \cdot n : n \in \mathbb{N}\}$. Then, the natural S-extension of L is not in $\mathtt{i.o.}\text{-}\mathsf{avg}_{\rho'}\text{-}\mathcal{BPTIME}[T']$, where $T' = T(n/2c)/\tilde{O}(t(n) \cdot d(n/2c) \cdot \ell(n/2c))$ and $\rho'(n) = \rho(\lfloor n/c \rfloor)$.*

Proof. Let $\bar{L}$ be the natural S-extension of L, and assume towards a contradiction that $\bar{L} \in \mathtt{i.o.}\text{-}\mathsf{avg}_{\rho'}\text{-}\mathcal{BPTIME}[T']$. Let M' be a probabilistic T'-time machine and let $S' \subseteq \mathbb{N}$ be an infinite set of input lengths on which M' solves $\bar{L}$ with average-case success ρ'. As in the proof of Lemma A.1, there are infinitely many $n \in S$ such that $I_n \cap S' \neq \emptyset$, where $I_n = [n, n+c)$.

We define M that solves $L^{(1)}$ infinitely often as follows. The machine M gets input $x_0 \in \{0,1\}^{n_0}$, and we assume that n_0 is such that $I_{c \cdot n_0} \cap S' \neq \emptyset$. Let Red be the linear-time reduction of deciding $L^{(1)}$ at inputs of length n_0 to deciding L at inputs of length $n = c \cdot n_0$ (such a reduction exists because the code is systematic). We run the instance checker for $O_c(\ell)$ times with input x_0, and in iteration $(i, a, r) \in \{0, ..., c-1\} \times [\ell] \times \{0,1\}^i$ we give it access to an oracle that is simulated as follows.

The oracle is given query $q \in \{0,1\}^{n_0}$, and it computes $q' = \mathsf{Red}(q)$. It then runs the local list-decoder Dec with input q', with index $a \in [\ell]$, and with oracle access to the following function: Whenever Dec queries $z \in \{0,1\}^n$, the oracle answers $M'(zr)$.

The machine M returns answer $\sigma \in \{0,1\}$ if and only if for some (i, a, r) the instance checker outputs σ, and for all $(i', a', r') \neq (i, a, r)$ the instance checker outputs a value in $\{\perp, \sigma\}$. Otherwise, the machine M outputs $\perp$.

Analysis. Without loss of generality (using naive error reduction) we assume that the errors of the instance checker and of Dec and of M are less than $\frac{1}{O(t(n_0) \cdot d(2n) \cdot \ell)}$, and let us condition on the high-probability event in which all executions of these three algorithms are correct.

By our assumption, for some $i \in \{0, ..., c-1\}$ it holds that M' computes $\bar{L}$ on at least $\rho = \rho(n_0)$ of the inputs of length $n + i$. Thus, for some $r \in \{0,1\}^i$ it holds that $\Pr_z[M'(zr) = L(z)] \geq \rho$. It follows that there exists a such that when Dec gets index a and oracle access to the function $z \mapsto M'(zr)$ it computes $\bar{L}$ on inputs of length $n + i$; that is, on query $q' = \mathsf{Red}(q)$ it returns $L^{(1)}(q)$. For the corresponding invocation of the instance checker, it will output the correct answer. Also, for all other invocations, it will output either the correct answer or $\perp$.

The running time of M is:

$$\begin{aligned} &\mathrm{polylog}(d(2n), t(n_0), \ell(2n)) \cdot t(n_0) \cdot d(c \cdot n_0 + c) \cdot T'(c \cdot n_0 + c) \\ &\leq \tilde{O}(t(n_0) \cdot d(2c \cdot n_0) \cdot \ell(2c \cdot n_0)) \cdot T'(2c \cdot n_0) \end{aligned}$$

which is at most $T(n_0)$, a contradiction. ■

The next claim is of the same spirit as Lemma A.4, but refers to an inherently different parameter setting. Specifically, we start from a function f that is mildly hard on average on almost all input lengths, and argue that a direct-product encoding $f^{\otimes k}$ of f is very hard on average on almost all input length. The assumption that we will use is that f has a tolerant instance checker, which implies that $f^{\otimes k}$ has a tolerant instance checker tolerating more corruption (see Claim 3.12.1).

Lemma A.5 (amplifying average-case hardness on all input lengths using a direct product and an instance checker)**.** *For every $c_1, c_2, c_3, c_4 \geq 1$ there exists a function $k(n) = \Theta(\log(n^{c_3 \cdot c_4}) \cdot n^{3c_3})$ such that following holds. Let $f \notin \texttt{i.o.-avg}_{(1-n^{-c_3})}\text{-}\mathcal{BPTIME}[n^{c_1}]$ that has a $(n^{-3c_3}, 10n^{-3c_3})$-tolerant same-length instance checker with running time n^{c_2}, and let $f^{\otimes k}$ be the k-wise direct product of f. Then, for $S = \{k \cdot n_0 : n_0 \in N\}$, the natural S-extension of $f^{\otimes k}$ is not in $\texttt{i.o.-avg}_{n^{-c_4}}\text{-}\mathcal{BPTIME}[n^c]$, where $c = \alpha \cdot \frac{c_1}{c_2 \cdot c_3 \cdot c_4}$ for a universal constant $\alpha > 0$.*

Proof. Let $\bar{f}$ be the natural S-extension of $f^{\otimes k}$, and assume that there exists an algorithm A running in time N^c on inputs of length N and an infinite set $S \subseteq \mathbb{N}$ such that for every $N \in S$ it holds that $\Pr_{z \in \{0,1\}^N}[A(z) = \bar{f}(z)] \geq N^{-c_4}$.

The proof is based on two main claims. The first claim is that we can use the success of A on infinitely many input lengths of arbitrary form to obtain an algorithm that succeeds on infinitely many input lengths of the form $k \cdot n$; this relies on the existence of a tolerant instance checker for f, which yields a tolerant instance checker for $f^{\otimes}$. In more detail:

Claim A.5.1. *There exists a universal constant $C' > 1$ and a probabilistic algorithm A' that runs in time $n^{C' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c}$ on inputs of length $k \cdot n$ and an infinite set $S' \subseteq \mathbb{N}$ such that for every $n \in S'$, the probability over $x_1, ..., x_k \in (\{0,1\}^n)^k$ that $\Pr[A'(x_1, ..., x_k) = f^{\otimes k}(x_1, ..., x_k)] \geq 2/3$ is at least $n^{-C' \cdot c_3 \cdot c_4}$.*

Proof. By Claim 3.12.1 with parameter values $\epsilon = n^{-(3c_3+2)\cdot(c_4+1)}$ and $\delta = n^{-3c_3}$ and $\delta' = 10n^{-3c_3}$, and relying on a sufficiently large choice of $k = \Omega(\log(1/\epsilon)/\delta)$, the problem $f^{\otimes k}$ has a $(1-\epsilon, 1-\epsilon')$-tolerant instance checker that runs in time at most $n^{C \cdot c_2 \cdot c_3 \cdot c_4}$, where

$$\epsilon' = (1 - 200n^{-3c_3})^k = (1 - 200n^{-3c_3})^{\Theta(\log(n^{c_3 \cdot c_4}) \cdot n^{3c_3})} = n^{-C \cdot c_3 \cdot c_4} ,$$

and $C > 1$ is a sufficiently large universal constant.

Let M' be the tolerant instance checker for $f^{\otimes k}$ above, and for every $n \in \mathbb{N}$ let $I_n = \{k \cdot n, k \cdot n + 1, ..., k \cdot n + k - 1\}$. The algorithm A' gets input $z \in \{0,1\}^{k \cdot n}$, and we assume that n is such that $I_n \cap S' \neq \emptyset$. Then, it repeats the following procedure for $t = O(N^{c_4+2}/\epsilon')$ times:

> Interval Attempt: For $i = 0, ..., k-1$, simulate M' with input z while answering the queries of M' as follows. First choose a random string $u_i \in \{0,1\}^i$ (which will be used for all queries in simulation i), and given any query $q \in \{0,1\}^{k \cdot n}$, run the algorithm A on input qu_i and return its output as answer to M'. If for some i the instance checker outputs $\sigma \in \{0,1\}^*$ (rather than $\perp$), abort and output σ; otherwise, continue.

If in all t repetitions of the Interval Attempt procedure above, the algorithm did not abort (i.e., in each interval attempt and for each $i \in [k]$ the instance checker returned $\perp$), the algorithm A' returns some default value (say, 0).

We assume that the error of M' is reduced to 2^{-n} (by naive error reduction for $O(n)$ times). Thus, with high probability all the outputs of M' will be in the set $\{f^{\otimes k}(z), \perp\}$. Now, let $i \in \{0, ..., k-1\}$ such that $N = k \cdot n + i \in S$. By our assumption it holds that $\Pr_{z' \in \{0,1\}^N}[A(z') = \bar{f}(z')] \geq N^{-c_4}$. We think of $z' \in \{0,1\}^N$ as $z' = (q, u_i)$ where $|q| = |z| = k \cdot n$ and $|u_i| = i$, and note that with probability at least N^{-c_4-1} over choice of $u_i \in \{0,1\}^i$ it holds that $\Pr_q[A(qu_i) = \bar{f}(qu_i)] \geq N^{-c_4-1}$. In this case, the function $O_{u_i}(q) = A(qu_i)$ agrees with $f^{\otimes k}$ on more than ϵ of the inputs,[44] and hence there exists a set $S_{u_i} \subseteq \{0,1\}^{k \cdot n}$ of density ϵ' such that for every $z \in S_{u_i}$ we have that $\Pr[(M')^{O_{u_i}}(z) = f^{\otimes k}(z)] \geq 2/3$. We call such choice of u_i good.

Now, consider a random choice of $z \in \{0,1\}^{k \cdot n}$ and of $u_i \in \{0,1\}^i$ and of coins for M'. Then, we have that

$$\begin{aligned}
&\mathbb{E}_z \left[\Pr_{u_i, \text{ coins for } M'} \left[(M')^{O_{u_i}}(z) = f^{\otimes k}(z) \right] \right] \\
&= \Pr_{z, u_i, \text{ coins for } M'} \left[(M')^{O_{u_i}}(z) = f^{\otimes k}(z) \right] \\
&\geq \Pr_{z, u_i} [u_i \text{ good} \wedge z \in S_{i_i}] \cdot \Pr_{z, u_i, \text{ coins for } M'} \left[(M')^{O_{u_i}}(z) = f^{\otimes k}(z) | u_i \text{ good} \wedge z \in S_{i_i} \right] \\
&\geq N^{-c_4-1} \cdot \epsilon' \,,
\end{aligned}$$

and hence there exists a set $Z \subseteq \{0,1\}^{k \cdot n}$ of density $N^{-c_4-2} \cdot \epsilon'$ such that for each $z \in Z$ it holds that $\Pr_{u_i, \text{ coins for } M'} \left[(M')^{O_{u_i}}(z) = f^{\otimes k}(z) \right] \geq N^{-c_4-2} \cdot \epsilon'$. By our choice of $t = O(N^{c_4+2} \cdot \epsilon')$, given any such z, with high probability A' outputs $f^{\otimes k}$.

The running time of A' is at most

$$\begin{aligned}
t \cdot k \cdot n^{c_2+1} \cdot N^c &\leq O\Big((k \cdot n)^{c_4+2} \cdot n^{C \cdot c_3 \cdot c_4} \cdot k \cdot n^{c_2+1} \cdot (k \cdot n)^c \Big) \\
&\leq \tilde{O}\Big(n^{(c_3+1) \cdot (c_4+2)} \cdot n^{C \cdot c_3 \cdot c_4} \cdot n^{c_3} \cdot n^{c_2+1} \cdot n^{c \cdot (c_3+1)} \Big) \\
&< n^{C' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c}
\end{aligned}$$

where $C' > C$ is a sufficiently large universal constant. □

The second main claim is that we can use an algorithm that computes $f^{\otimes k}$ on infinitely many input lengths of the form $k \cdot n$ to obtain an algorithm that computes f on infinitely many input lengths. The proof combines the instance checker for f with the approximate local decoder from Theorem 3.10.

Claim A.5.2. *There exists a universal constant $C'' > 1$ and a probabilistic algorithm A'' that runs in time $n^{C'' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c}$ and an infinite set $S'' \subseteq \mathbb{N}$ such that for every $n \in S''$, with probability at least $1 - n^{-c_3}$ over $x \in \{0,1\}^n$ it holds that $\Pr[A''(x) = f(x)] \geq 2/3$.*

Proof. Let $\delta = n^{-3c_3}$ and $\epsilon = n^{-2C' \cdot c_3 \cdot c_4}$, and let M be the $(\delta, 10\delta)$-tolerant instance checker for f. Given input $x \in \{0,1\}^n$, the algorithm A'' repeats the following procedure for $t = \Theta(\log(1/\delta)/\epsilon^2)$ times:

1. Choose a random string u_i of length $n^{C' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c}$, and let $O_{u_i} \colon \{0,1\}^{k \cdot n} \to \{0,1\}^*$ such that $O_{u_i}(z)$ is the output of A' on z when the random coins of A' are fixed to be u_i.

[44] Since $\epsilon = n^{-(3c_3+2) \cdot (c_4+1)}$ whereas $N < (n+1) \cdot k < n^{3c_3+2}$, which implies that $\epsilon < N^{-c_4-1}$.

2. Consider Theorem 3.10, instantiated with parameter values k, ϵ, δ (note that our choice of k is indeed sufficiently large). Run the $\mathcal{NC}^0$ decoder with oracle access to O_{u_i} to obtain a circuit C_{u_i} of size $\text{poly}(n, k, \log(1/\delta), \epsilon)$ with oracle gates to O_{u_i}, and hard-wire O_{u_i} in place of these oracle gates.

3. Run M on input x, its error reduced to 2^{-n}, while answering its queries according using C_{u_i}. If M outputs a value $\sigma \in \{0,1\}^*$ different than $\perp$, output σ and abort.

If M outputs $\perp$ in all t attempts M, the algorithm A'' outputs the default value 0.

Turning to the analysis, observe that with high probability, the instance checker M will only output values in the set $\{f(x), \perp\}$; thus, we only need to prove that with high probability, in some iteration M will output $f(x)$.

By the properties of A' we have that $\Pr_{x_1,\ldots,x_k, \text{ coins for } A'}[A'(x_1, \ldots, x_k) = f^{\otimes k}] \geq (2/3) \cdot n^{-C' \cdot c_3 \cdot c_4} = (2/3) \cdot \sqrt{\epsilon} > 2\epsilon$. Hence, with probability at least ϵ over choice of random coins u_i for A' it holds that $\Pr[O_{u_i}(z) = f^{\otimes z}] \geq \epsilon$. Whenever this happens, by Theorem 3.10, with probability $\Omega(\epsilon)$ over random coins r_i for the $\mathcal{NC}^0$ decoder it holds that $\Pr_{q \in \{0,1\}^n}[C_{u_i}(x) = f(x)] \geq 1 - \delta$. Denote by $\mathcal{E}_i$ the event that $\Pr_{q \in \{0,1\}^n}[C_{u_i}(x) = f(x)] \geq 1 - \delta$ (this event depends on r_i and u_i); whenever $\mathcal{E}_i$ happens, by the properties of M, there exists a set $X_{u_i} \subseteq \{0,1\}^n$ of density at least $1 - 10\delta$ such that for every $x \in X_{u_i}$ it holds that $\Pr[M^{C_{u_i}}(x) = f(x)] \geq 2/3$.

Since $t = \Theta(\log(1/\delta)/\epsilon^2)$, the probability that $\mathcal{E}_i$ happens for some $i \in [t]$ is at least $1 - \delta$. Thus, the probability over $x \in \{0,1\}^n$ and $(u_i, r_i)_{i \in [t]}$ that $\Pr[M^{C_{u_i}}(x) = f(x)] \geq 2/3$ for some i is at least $1 - 2\delta$. It follows that there exists a set $X_0 \subseteq \{0,1\}^n$ of density $1 - \sqrt{2\delta}$ such that for every $x \in X_0$, with probability at least $1 - \sqrt{2\delta} > 1 - n^{-c_3}$ over $(u_i, r_i)_{i \in [t]}$ it holds that $\Pr[M^{C_{u_i}}(x) = f(x)] \geq 2/3$ for some i.

For every $x \in X_0$, with probability at least 0.6 the algorithm A'' outputs $f(x)$ (where the difference between $2/3$ and 0.6 accounts for conditioning on the fact that M outputs values in $\{\perp, f(x)\}$ in all iterations, and on the probability $1 - \sqrt{2\delta}$ over $(u_i, r_i)_{i \in [t]}$). We can increase the probability of success to $2/3$ by naive error-reduction. The running time of A'' is

$$t \cdot \text{poly}(n, k, \log(1/\delta)) \cdot n^{C' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c} \cdot n^{c_2} < n^{C'' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c},$$

for a sufficiently large universal constant $C'' > 1$. □

Setting $\alpha = 1/C''$ we have that $C'' \cdot c_2 \cdot c_3 \cdot c_4 \cdot c < c_1$, and hence the algorithm A'' above contradicts the hypothesis that $f \notin \texttt{i.o.-avg}_{(1-n^{-c_3})}\text{-}\mathcal{BPTIME}[n^{c_1}]$. ■

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il