

Trading Time, Space, and Alternations: General Plasticity for Algorithms from Hardness of Range Avoidance

Zeyong Li

Roei Tell

October 7, 2026

Abstract

In a recent breakthrough, Williams (STOC 2025) showed that any decision problem solvable in time t by a multitape Turing machine can be solved in space $\tilde{O}(\sqrt{t})$ and time $2^{\tilde{O}(\sqrt{t})}$. The immediate question is whether this result is an anomaly, or an inherent feature of computation more generally.

In this work we provide evidence that the resource trade-offs phenomenon first demonstrated by Williams is systematic, i.e. inherent to computation in general. Specifically, under a plausible complexity-theoretic hardness assumption, we show that t -time algorithms can be simulated in space t^ε , for any constant $\varepsilon > 0$, where the simulation is correct on average over a random input. We also show, under similar complexity-theoretic hardness assumptions, that t -time algorithms can be simulated in *linear time* using sufficiently many alternations (i.e., $\forall/\exists$ quantifiers), where again the simulation is correct on average over a random input.

Some hardness assumption is necessary to prove these conclusions (as they imply that $P \neq PSPACE$), and the specific hardness assumption that we rely on has been extensively studied in complexity theory in recent years: hardness of a computational problem called Range Avoidance. Specifically, we introduce a new “low-space” variant of Range Avoidance, and show (under mild derandomization assumptions) that average-case hardness of this problem for polynomial-time algorithms is in fact *equivalent* to low-space simulation of FP, and implies linear-time simulation with alternations. We then study the complexity of this new problem, showing conditional hardness results and algorithms.

Contents

1	Introduction	1
1.1	Hardness of Range Avoidance	2
1.2	Plasticity for algorithms from hardness of Avoid	3
1.3	On the complexity of low-space Avoid	4
1.4	A technical tangent: targeted hitting-set generators for <i>coNP</i>	5
1.5	Discussion	5
2	Technical Overview	7
2.1	Worst-case hardness of low-space Avoid from $\text{NP} \neq \text{PSPACE}$	7
2.2	Plasticity from average-case hardness of low-space Avoid	8
2.3	A non-deterministic algorithm for low-space Avoid: Proof of Theorem 1.6	11
2.4	Hardness of Avoid for tiny-error non-deterministic algorithms: Proof of Theorem 1.8	12
2.5	A targeted hitting-set generator for a subclass of <i>coNP</i>	13
3	Preliminaries	13
4	PCP for Sublinear Time	17
4.1	Known technical tools	17
4.2	Proof of Theorem 4.1	18
5	Plasticity for Algorithms from Hardness of Low-Space Avoid	21
5.1	Simulating polytime by a low-space <i>AM</i> verifier	21
5.2	Consequences I: Trading time for space and alternations for time	24
5.3	Consequences II: An equivalence	27
6	The Complexity of Low-Space Avoid	31
6.1	Worst-case hardness for non-deterministic algorithms	31
6.2	Average-case hardness for tiny-error non-deterministic algorithms	33
6.3	An average-case non-deterministic algorithm	37
7	Targeted Hitting-Set Generators for <i>coNP</i> (and Avoid)	39
7.1	There is no targeted generator with few outputs hitting <i>coNP</i> in the worst-case	40
7.2	A targeted generator with few outputs hitting a subclass of <i>coAM</i> on average	41
A	Prior Assumptions that Do Not Coincide with Low-Space Simulation of FP	51

1 Introduction

In a recent breakthrough, Williams [Wil25] showed that any decision problem solvable in time t by a multitape Turing machine can be solved in space $\tilde{O}(\sqrt{t})$ (and time $2^{\tilde{O}(\sqrt{t})}$). His low-space algorithm relies on a low-space algorithm by Cook and Mertz [CM24] for a specific problem called Tree Evaluation.

This result is surprising, and we immediately wonder why this low-space simulation is possible. One possibility is that it is an anomaly in the computational landscape. For example, that we can only simulate a particular computational model in such space (i.e., multitape machines), but simulations of other natural models (e.g., RAM machines) are weaker.¹ Alternatively, perhaps we can only save a quadratic factor in space (or another fixed polynomial factor), but no more, for reasons that are yet to be understood.

Concretely, to understand the space savings phenomenon and delineate its scope, the following questions naturally arise, and most of them were posed in [Wil25]:

1. Can we simulate t -time RAM machines in space $\tilde{O}(\sqrt{t})$? Going further, can we simulate t -time machines in space t^ε for an arbitrarily small $\varepsilon > 0$?
2. Is it possible to trade resources other than time and space? For example, can we simulate time t in non-deterministic time $t^{1-\Omega(1)}$, or higher in the polynomial-time hierarchy (i.e., using constantly many $\exists/\forall$ quantifiers) in time $t^{1-\Omega(1)}$?
3. As observed in [Wil25], simulating P in space n^ε (for an arbitrarily small $\varepsilon > 0$) implies that $P \neq PSPACE$. Following Kozen’s [Koz78] classical conjecture that lower bounds can be proved by simulation and diagonalization, can we provide evidence that $P \neq PSPACE$ is indeed provable in this manner?

Our contributions. We provide a *conditional positive answer to all three questions above*. At a high level, under a plausible complexity-theoretic hardness assumption, we show that t -time machines can be simulated in space t^ε , for any constant $\varepsilon > 0$, where the simulation is correct on average over a random input. We also show, under similar complexity-theoretic hardness assumptions, that t -time machines can be simulated in *linear time* using sufficiently many alternations (i.e., $\forall/\exists$ quantifiers), where again the simulation is correct on average over a random input. These average-case simulations suffice to imply that $P \neq PSPACE$.

As a main takeaway, our results provide evidence that *the phenomenon demonstrated in [Wil25] is systematic*, i.e. inherent to computation in general, rather than an anomaly that occurs only in some single specific setting.

Our results only provide evidence, rather than a conclusive answer, because they are conditioned on a hardness assumption. The specific assumption that our results rely on has been intensively studied in complexity theory in recent years, and it refers to the hardness of a problem called Range Avoidance.

Specifically, we introduce a new “low-space” variant of Range Avoidance, and show (under mild derandomization assumptions) that average-case hardness of this problem for polynomial-time algorithms is in fact *equivalent* to low-space simulation of P, and implies linear-time simulation with alternations. We then study the complexity of this new problem, showing conditional hardness results and algorithms.

Terminology: Resource trade-offs and algorithmic plasticity. What was referred to above as “space savings” is actually a resource trade-off: we can simulate any algorithm using less space, but more time. This contrasts with the belief (which the authors certainly held before [Wil25]) that certain algorithms are “rigid”, in the sense that one cannot significantly reduce their usage of *any* resource, even at the expense of other resources.

¹As pointed out by Ryan Williams, the recently claimed simulation of one-tape machines in space $\tilde{O}(t^{2/5})$ [Ope26] implies simulation of RAM machines in space $\tilde{O}(t^{4/5})$.

The phenomenon that was discovered in [Wil25], and for whose generality we provide evidence, points at the opposite of rigidity in general, and we thus suggest referring to this phenomenon as the *plasticity of algorithms* (or “algorithmic plasticity”). For example, our results suggest that for any algorithm, one can “shrink” its space while “stretching” its time to be sub-exponential; at least, this is true if our assumptions hold, and as long as one is willing to preserve the original functionality only on average over a random input.

1.1 Hardness of Range Avoidance

In the Range Avoidance problem (Avoid, in short), introduced by Kleinberg *et al.* [KKMP21], the input is a circuit $C: \{0, 1\}^\ell \rightarrow \{0, 1\}^k$ with $\ell < k$, and a solution to the problem is any k -bit string outside the range of C . Since $\ell < k$, a random string would be outside the range with high probability, and the main challenge is to solve this problem deterministically or pseudo-deterministically. This challenge received significant recent attention (see, e.g., the survey by Korten [Kor25]), motivated in part by Korten’s proof [Kor21] that solutions to Avoid – even for special cases – imply explicit constructions of long-sought objects, such as truth-tables hard for non-uniform circuits (AKA circuit lower bounds), Ramsey graphs, and rigid matrices.

There are several known algorithms for Avoid. Chen *et al.* [CHLR25] showed a pseudodeterministic FZPP^{NP} algorithm,² and Ren and Williams [RW26] showed an algorithm in $\text{FP}^{\text{prMA}}/1$; these indicate that Avoid may be solvable in FP^{NP} . Furthermore, Chen, Li, and Liang [CLL25] showed an algorithm in $\text{AMTIME}[2^{\text{polylog}(n)}]/n^{o(1)}$, and Ghentiyala, Li, and Stephens-Davidowitz [GLS26] showed that *decision problems* reducible to Avoid can be solved in $\text{AM} \cap \text{coAM}$; these indicate that Avoid may even be solved in complexity closer to FNP. More algorithms are known for special cases; see the survey [Kor25].

Alas, we also have evidence that Avoid may be a hard problem. Ilango, Li, and Williams [ILW23] showed that Avoid cannot be solved in polynomial time, under cryptographic and complexity-theoretic assumptions (subexponentially strong indistinguishability obfuscation and $\text{NP} \neq \text{coNP}$).³ Going further, Chen and Li [CL24] showed that Avoid cannot be solved in *non-deterministic* polynomial time, under strong cryptographic assumptions (a variant of LWE against non-deterministic adversaries, and an assumption they called “inhomogeneous short integer solution”; see [CL24, Theorem 1.11]). Further evidence of this was shown by Ren, Wang and Zhong [RWZ26].⁴ Making matters even worse, Ilango [Ila25] showed that Avoid is hard for zero-error non-deterministic polynomial time algorithm even *on average*, assuming that there are no NP -constructive natural properties. The meaning of “zero-error” is that the algorithm is never allowed to output an incorrect solution (but on faulty inputs it may output $\perp$ on all non-deterministic witnesses).

A low-space variant of Avoid. We will consider a generalized form of Avoid, which is implicit in the recent work of Ilango [Ila25]. In this problem, instead of receiving a circuit C_x , the algorithm only receives x , and needs to solve $\text{Avoid}(C_x)$ for C_x that is implicitly defined and that is computable from x . More formally, we consider indexed families of Avoid instances $C = \{C_x\}_{x \in \{0, 1\}^*}$, in particular uniform circuit families in which $C_x(r)$ is efficiently computable from x and r , and the problem Avoid_C whose hardness we consider is: given x , find a string outside the range of C_x . Intuitively, this decouples the non-uniformity (i.e., the string x) and the uniform algorithmic part (i.e., the algorithm computing $C_x(r)$ from x and r).

The variant that we consider is solving Avoid_C for any indexed family $C = \{C_x\}$ that is evaluable in low space, i.e. space n^ε for a small constant $\varepsilon > 0$. This variant allows very big circuits (of size 2^{n^ε}), but it enforces a low-space-evaluability constraint. Low-space Avoid is on the harder extreme of the spectrum of Avoid variants: for example, all known hardness results for Avoid mentioned above (when instantiated with low-space cryptographic primitives) yield special cases of this problem. We also provide stronger evidence for hardness: a worst-case lower bound for low-space Avoid against non-deterministic algorithms follows from the standard complexity assumption $\text{NP} \neq \text{PSPACE}$ (see Section 1.3).

²In fact, their algorithm is stronger and works in single-valued FS_2 (i.e., the search version of the symmetric class S_2).

³Their result applies to Avoid for polynomial-sized circuits, with polynomial stretch. For simplicity, we omit these parameters in this high-level descriptions.

⁴Indeed, the conditional hardness in [CLL25, RWZ26] stands in contrast to the algorithms of [CLL25, GLS26], which suggest that Avoid may be in FNP or close to that. The intuitive reason Avoid might be hard for non-deterministic algorithms is that it is not clear how to verify that a given $y \in \{0, 1\}^k$ is outside the range of an input circuit (this problem is in coNP).

The main assumption in our results is that low-space Avoid with polynomial stretch is hard for deterministic polynomial-time algorithms on average. Note that this assumption is incomparable to the conclusion in Ilango’s work [Ila25], which asserts average-case hardness for zero-error non-deterministic algorithms: on the one hand, we assume hardness only for *deterministic algorithms*, rather than non-deterministic ones; but on the other hand, the algorithms are allowed to err on “bad” inputs. In more detail:

Assumption 1.1. *For every sufficiently small constant $\varepsilon > 0$ there is a family of Avoid instances $C = \{C_x\}_{x \in \{0,1\}^*}$ stretching $|x|^{\varepsilon^2}$ bits to $|x|^\varepsilon$ bits and evaluable in space $|x|^{\varepsilon^2}$ such that for some $p: \mathbb{N} \rightarrow (0,1)$ the following holds.⁵ For every polynomial-time algorithm A and every large enough $n \in \mathbb{N}$, we have*

$$\Pr_{x \in \{0,1\}^n} [A(x) \text{ solves Avoid}(C_x)] \leq p(n) .$$

Note that the stretch in Assumption 1.1 is an arbitrarily large polynomial $(\cdot)^{1/\varepsilon}$, and that the output length of C_x is sub-linear in the index length $|x|^\varepsilon \ll |x|$. The sub-linear output length makes the assumption weaker (i.e., better), and we in fact suspect that for certain strong versions of the assumption (i.e., when C_x is also evaluable in polynomial time) are false when the output is super-linear (see Section 1.4).

Also note that the notion of “low-space” refers to space that is low compared to both the index $|x|$ and the output $|x|^\varepsilon$ of C_x . In particular, the space is smaller than the circuit size, since $|C_x| \geq |x|^\varepsilon$. A useful hard candidate to keep in mind is solving Avoid for logspace-uniform circuits C_x of depth n^{ε^2} and size $2^{n^{\varepsilon^2}}$.

1.2 Plasticity for algorithms from hardness of Avoid

The main results, which motivate the study of low-space Avoid throughout the rest of this paper, assert that the combination of Assumption 1.1 with derandomization assumptions yields algorithmic plasticity.

Our first result trades space for time. In fact, under the derandomization assumption, it shows that such a tradeoff is *equivalent* to hardness of low-space Avoid. To state the result, recall that prBP^*L , introduced by Nisan [Nis93], is the class of promise problems solvable by probabilistic machines running in logspace and polynomial time whose randomness is stored on a special tape to which they have persistent two-way read-only access (and the contents of this tape does not count towards the machine’s space complexity).⁶ The derandomization assumption is that such randomness is unhelpful, i.e. that $\text{prBP}^*\text{L} = \text{prL}$. This assumption follows from circuit lower bounds as well as other types of lower bounds (see [KvM02, PR23, Sha24]).

Theorem 1.2 (trading space for time). *If $\text{prBP}^*\text{L} = \text{prL}$, then the following two statements are equivalent:*

1. *Assumption 1.1 holds with $p(n) = n^{-\omega(1)}$.*
2. *For every $f \in \text{FP}$ and every $\varepsilon > 0$ there is a space- n^ε algorithm A such that $\Pr_{x \in \{0,1\}^n} [A(x) = f(x)] \geq 1 - n^{-\omega(1)}$ for all $n \in \mathbb{N}$.*

Our second result trades time for alternations. For this result we will assume hardness for a special case of low-space Avoid, in which C_x is evaluable in simultaneous time $\text{poly}(n)$ and space n^{ε^2} (indeed, this is also a special case of the standard formulation of Avoid). For a concrete candidate hard problem, one may consider a family $C = \{C_x\}$ of logspace-uniform NC^1 circuits,⁷ or highly sequential circuits of width n^ε and depth $\text{poly}(n)$. We will also need a stronger derandomization assumption; for simplicity, let us assume a circuit lower bound that suffices for our purposes and that is commonly assumed in complexity theory, although this assumption seems like an overkill (see Section 2).

⁵For our proofs it is important that C_x takes $|x|^{\varepsilon^2}$ input bits, but the proofs go through when the output length is larger than $|x|^\varepsilon$. Since Avoid is harder when the input length is smaller, we stated Assumption 1.1 with the smallest value that our proofs support (i.e., $|x|^\varepsilon$).

⁶That is, in contrast to standard probabilistic machines, which “toss coins” but need to use space on their worktape to record the outcomes, in this stronger model the outcomes are recorded “for free” on the special worktape.

⁷Recall that Chen and Li [CL24] deduced worst-case hardness of Avoid for NC^1 circuits (for non-deterministic algorithms) from cryptographic assumptions.

Theorem 1.3 (trading time for alternations). *Assume that $\text{DSPACE}[O(n)] \not\subseteq \text{i.o.SIZE}^{\text{SAT}}[2^{\delta n}]$ for some $\delta > 0$, and that Assumption 1.1 holds with $p(n) = n^{-\omega(1)}$ and $C = \{C_x\}$ that is evaluable in simultaneous time $\text{poly}(n)$ and space n^{ε^2} . Then, for every $L \in \mathbf{P}$ there is a constant $k = k_L$ and a linear-time Σ_k machine that decides L correctly on at least $1 - n^{-\omega(1)}$ fraction of the n -bit inputs.*

The parameter value $p(n) = n^{-\omega(1)}$ in Theorems 1.2 and 1.3 was chosen for simplicity (recall that in [Ila25] the bound $p(n)$ on solving Avoid was exponentially small); one can choose a different $p(n)$ in the assumption and it will carry to the conclusions. Also, the hardness assumption is over the uniform distribution and the simulation is also over the uniform distribution, but hardness over other distributions yields simulation over a corresponding distribution. See Theorem 5.4 for details.

Results for polynomial time can scale up to larger times, using an algorithm that adds random padding to the input. In our case, for *exponential-time* algorithms we can do better: the same assumptions imply a *worst-case* simulation (rather than only on average). This is true both for the low-space simulation and for the linear-time Σ_k simulation.

Theorem 1.4 (worst-case plasticity for exponential time). *Under the assumptions of Theorem 1.2,*

$$E \subseteq \cap_{\varepsilon > 0} \text{DSPACE}[2^{\varepsilon \cdot n}] .$$

and under the assumptions of Theorem 1.3,

$$E \subseteq \cap_{\varepsilon > 0} \cup_k \Sigma_k \text{TIME}[2^{\varepsilon \cdot n}] .$$

The point underlying Theorem 1.4 is that the gap between average-case simulation and worst-case simulation can be bridged for exponential-time algorithms, by a standard worst-case to average-case reduction.

1.3 On the complexity of low-space Avoid

Given these results, the complexity of low-space Avoid is a major open problem. We turn our attention to it.

Our first result is conditional worst-case hardness for non-deterministic algorithms. Indeed, such hardness was already deduced in prior works [ILW23, CL24, Ila25, RWZ26], but these works relied on strong cryptographic assumptions, whereas we deduce it from a standard complexity-theoretic assumption. We view this stronger evidence as indication that low-space Avoid may be harder than previously-studied variants of Avoid.

Theorem 1.5 (worst-case hardness for non-deterministic algorithms). *If $\text{NP} \neq \text{PSPACE}$ then for every $\varepsilon > 0$ there is a family $C = \{C_x : [4] \rightarrow \{0, 1\}^{|x|^\varepsilon}\}$ evaluable in space $|x|^{\varepsilon^2}$ such that Avoid_C is not solvable in non-deterministic polynomial time (in the worst-case).*

Our next result concerns the *average-case* complexity of low-space Avoid for *non-deterministic algorithms*. One may be tempted to strengthen both the assumptions in Section 1.1 (which assert average-case hardness for deterministic algorithms) and the conclusion in [Ila25] (which asserts average-case hardness for zero-error non-deterministic algorithms), and simply assume that low-space Avoid is average-case hard for non-deterministic algorithms that are allowed to err. We provide strong evidence that such an assumption is false.

Theorem 1.6 (an average-case non-deterministic algorithm). *Assume that there are sub-exponentially secure one-way functions. Then, for some constant $\varepsilon > 0$ there is a non-deterministic polynomial-time algorithm A using at most n^ε non-deterministic bits such that the following holds. For every family of Avoid instances $C = \{C_x\}_{x \in \{0,1\}^*}$ stretching $|x|^{\varepsilon^2}$ bits to $|x|^\varepsilon$ bits and evaluable in time $2^{|x|^{\varepsilon^2}}$, for infinitely many $n \in \mathbb{N}$,*

$$\Pr_{x \in \{0,1\}^n} [A(x) \text{ solves } \text{Avoid}(C_x)] \geq 1/\text{poly}(n) ,$$

where the meaning of “solves $\text{Avoid}(C_x)$ ” is that on some witness $A(x)$ outputs an Avoid solution, and for every witness it outputs either an Avoid solution or $\perp$.

The success probability of $1/\text{poly}(n)$ is modest. Under stronger cryptographic assumptions (e.g., sub-exponentially secure one-way permutations) it can be increased to $1 - n^{-\omega(1)}$; see [Section 6.3](#).

Remark 1.7. *Theorem 1.6 may seem surprising, and it calls into question the original intuition that Avoid should be hard for deterministic algorithms on average, as in Assumption 1.1. However, there seems to be a reason for this. In Section 1.5 we explain why it makes sense for Avoid to be hard for deterministic algorithms and easy for non-deterministic algorithms (on average).*

At a superficial glance, the difference between the conclusion in [\[Ila25\]](#) and [Theorem 1.6](#) seems to be the “zero-error” restriction. Thus, one may be tempted to assume that zero-error delineates the difference between easiness and hardness (on average) for non-deterministic algorithms. In our last result concerning the complexity of low-space Avoid, we provide evidence that the feature of having literally *zero* error is not crucial for hardness. Specifically, we strengthen the conclusion from [\[Ila25\]](#) by extending the average-case hardness to hold for non-deterministic algorithms that are allowed to err on a very small fraction of inputs.

Theorem 1.8 (average-case hardness for tiny-error non-deterministic algorithms). *Assume that there is a demi-bits generator secure against AM and computable in space $s(n) \geq \log(n)$.⁸ Then, there is an family $C = \{C_x\}$ evaluable in space $O(s)$ such that every non-deterministic polynomial-time machine for Avoid_C that errs on at most $2^{-2\sqrt{n}}$ of the n -bit inputs succeeds on less than $O(2^{-\sqrt{n}})$ of the inputs.*

The full technical statement of [Theorem 1.8](#) supports a tradeoff between the allowed error and the bound on the success probability; see [Theorem 6.6](#) and [Corollary 6.7](#).

1.4 A technical tangent: targeted hitting-set generators for coNP

A natural strategy for solving low-space Avoid (as well as other variants of Avoid) is to use a targeted hitting-set generator, as defined by Goldreich [\[Gol11a\]](#). Such a generator gets input x , and produces a list of strings $w_1, \dots, w_L$ such that, for every procedure $D(x, \cdot)$ that belongs to a target class $\mathcal{D}$ and satisfies $\Pr_r[D(x, r) = 1] \geq 1/2$, there is w_i for which $D(x, w_i) = 1$. In the case of Avoid, we consider $\mathcal{D} \subseteq \text{coNP}$, and we want to combine the outputs into a single solution. We know how to do so when the number of output strings is small, i.e. $L \ll n$ (e.g., by concatenating $y = w_1 \circ \dots \circ w_L$, as in [\[KS25\]](#); see [Section 7](#)).

The question of whether there are efficiently computable targeted hitting-set generators for coNP , even under the assumption that $\text{AM} = \text{NP}$, is an open problem. (Since such generators imply $\text{AM} = \text{NP}$, the open problem essentially asks whether $\text{AM} = \text{NP}$ is equivalent to the existence of such generators.) This problem was first posed by Goldreich [\[Gol11a\]](#), and has been studied in recent works [\[vMS23a, vMS23b\]](#). There is evidence that such generators do exist (see [\[IKW02, vMS23a\]](#)), but these have a large output-set $L = \text{poly}(n)$.

Loosely speaking, we prove results indicating that the complexity of computing a hitting-set generator for coNP with a small output-set $L \ll n$ is analogous to the complexity of solving low-space Avoid (i.e., to the results from [Section 1.3](#)).⁹ At a high level, we show that there are *no such generators* that work in the worst-case and are computable in non-deterministic polynomial time, assuming that Avoid is worst-case hard for non-deterministic polynomial time; and that there are such generators computable in non-deterministic polynomial time that fool a subclass of coNP on average over a random input (the subclass consists of procedures that use few co-non-deterministic bits), assuming sub-exponentially secure one-way functions. Our results do not shed any light about whether or not there are such generators that are computable in *deterministic* polynomial time and work on average. For brevity, we postpone precise details to [Section 7](#).

1.5 Discussion

In [Figure 1](#) we summarize the results about the complexity of low-space Avoid. The reader may notice that we have evidence (in most cases strong evidence) for its complexity in *all regimes except the most important one*, which is average-case hardness for deterministic algorithms.

⁸See [Definition 6.3](#).

⁹Prior work on this open problem did not distinguish between this case and the general case. We also note that this special case is of particular importance to superfast derandomization. See [Section 7](#) for details.

	Average-case	Worst-case
Deterministic	Hardness equivalent to low-space simulations (Thm 1.2)	Hard, assuming $\text{NP} \neq \text{PSPACE}$ (Thm 1.5)
Tiny-error non-deterministic	Hard, assuming demi-bits against AM (Thm 1.8)	
General non-deterministic	Easy, assuming subexp-OWFs (Thm 1.6)	

Figure 1: The complexity of low-space Avoid for polynomial-time algorithms. The cell highlighted in green indicates the most important regime, which we understand the least, and this cell assumes $\text{prBP}^*\text{L} = \text{prL}$.

As mentioned in the paper’s opening, we view our results as providing evidence that Williams’ [Wil25] result is an instance of more general algorithmic plasticity. The point is not simply deducing plasticity from some plausible hardness assumption. (It is easy to show that there exists an artificial low-space function f^* such that low-space simulation of FP is equivalent to f^* being hard for FP; see Section 5.3.) The point is basing plasticity on hardness of an *an interesting and seemingly difficult* problem, and moreover on a seemingly *extra-hard variant*. Indeed, Avoid is a general computational problem (e.g., the foregoing f^* reduces to it), and more importantly, a natural problem whose hardness has been intensively explored recently for its inherent interest. Moreover, since we show that hardness of low-space Avoid is also necessary for low-space simulation of FP (assuming $\text{prBP}^*\text{L} = \text{prL}$), we closely tie together these two important lines of research.

If one indeed conjectures that low-space Avoid is hard on average for polynomial-time algorithms, one has to contend with the fact that it is easy on average for *non-deterministic algorithms* (assuming sub-exponentially strong OWFs). We argue that this situation is actually coherent and makes sense. Our main technical intuition is thinking of an algorithm for low-space Avoid as solving a problem harder than the problem “compute a function hard for low-space algorithms”. To see this, consider a low-space M outputting $k = k(n)$ bits, and let $\mathcal{C} = \{C_x\}_x$ be the family with stretch $0 \mapsto k$ such that $C_x(\lambda) = M(x)$ (where λ denotes the empty string). Then, on input x , the Avoid algorithm outputs a string $z \neq M(x)$.¹⁰ Indeed, an Avoid algorithm solves a task that seems *significantly harder* than computing a hard function, because it also works for (the more challenging situation of) circuits with smaller stretch $\ell \mapsto k$.

How is this intuition informative? If there is a low-space simulation of FP as in Theorem 1.2, then there is no function in FP hard for space n^ε ; in particular, there can be no FP algorithm for low-space Avoid. (Indeed, this is the crux of the easier direction in the proof of Theorem 1.2.) However, assuming sub-exponential OWFs, there *is* a function in NP that is hard for low space on average. This opens the door for a non-deterministic average-case algorithm for low-space Avoid, as in the conclusion of Theorem 1.6.

¹⁰In this description, the Avoid algorithm for the family $\{C_x\}$ diagonalizes only against one algorithm M that specifies C_x . However, we can consider a universal family $\{U_x\}$ that parses a small prefix x^{pre} of x as a description of a low-space machine M , and evaluates M on the suffix x^{suf} of x . An algorithm that solves Avoid for $\{U_x\}$ on the input-set $\{x^{\text{pre}}x^{\text{suf}}\}_{x^{\text{pre}}}$ diagonalizes simultaneously against all low-space machines on x^{suf} .

Does algorithmic plasticity make sense? Low-space simulation of FP is remarkably consistent with known results and conjectures in theoretical computer science. As far as we are aware, the main implication is that assumptions made in a small number of works are false. The conclusions in these works – often derandomization conclusions – may still be true. The same applies to simulation of FP in linear-time with alternations. See Appendix A for the few examples we are aware of.

Low-space simulations also align with Kozen’s [Koz78] conjecture in the context of PSPACE vs P. Recall that his conjecture essentially says that lower bounds can always be proved via diagonalization; that is, if a class $\mathcal{C}$ has a hard function for a class $\mathcal{D}$, then $\mathcal{C}$ can simulate $\mathcal{D}$ (and diagonalize, assuming $\mathcal{C}$ is closed under complementation; see also [NIR03, NIR06, CT21a]). In our context, the conjecture means that $\text{PSPACE} \neq \text{P}$ can be proved by low-space simulations, and our results strengthen the evidence that this may be possible. (Indeed, the average-case simulations in our results imply that $\text{P} \neq \text{PSPACE}$; see Theorem 3.13.)

Three technically related works. Theorem 1.6 is similar to results by Korten [Kor22] and by Korten and Santhanam [KS25], since all three deduce algorithms for Avoid from hardness assumptions. In [Kor22, KS25] they focused on algorithms for the important special case of *uniform* Avoid, which in our terminology is Avoid_C for an *unindexed* circuit family $C = \{C_n\}_{n \in \mathbb{N}}$ (i.e., one circuit C_n for every input length $n \in \mathbb{N}$).

Specifically, Korten [Kor22] showed that if non-deterministic exponential time cannot be simulated in simultaneous low space and slightly more time, then uniform Avoid can be solved in FP^{NP} . And Korten and Santhanam [KS25] showed that uniform Avoid is in P, under strong (but reasonable) derandomization assumptions. In comparison, our algorithm for Avoid is non-deterministic, works for indexed families and on a noticeable fraction of inputs, and relies on a widely believed cryptographic assumption.

In addition, as pointed out by Ryan Williams, Theorem 1.2 is reminiscent of a result of Santhanam [San23]. Similarly to our result, in [San23, Theorem 4] it is proved, under a derandomization assumption, that hardness of low space is equivalent to an algorithmic task being carried out in low space. However, in contrast to our result, in [San23] the hardness is for any problem in PSPACE, and the algorithmic task is specific (i.e., low-space sampling of satisfying assignments for an implicitly given circuit; see [San23, Corollary 41]).

2 Technical Overview

It will be most convenient to first present the proof of Theorem 1.5, which is self-contained, and we do so in Section 2.1. Then we will describe the proofs of the results from Section 1.2 and the proof of Theorem 1.6 (which is, technically, an extension of the foregoing proofs), in Section 2.2 and Section 2.3, respectively. We will describe the proof of Theorem 1.8 in Section 2.4. And finally, in Section 2.5 we will give a very brief overview of the construction and analysis of the generator mentioned in Section 1.4.

2.1 Worst-case hardness of low-space Avoid from $\text{NP} \neq \text{PSPACE}$

Assuming that for some $\varepsilon > 0$ and every low-space $C = \{C_x\}$ there is a non-deterministic algorithm for Avoid_C , we will construct an NP verifier V for the PSPACE-complete problem TQBF. The construction of V follows an idea of Buhrman and Torenvliet [BT00].

The NP verifier V gets a totally quantified formula φ , and considers a suitable version of the $\text{IP} = \text{PSPACE}$ protocol, executed on input φ . It non-deterministically guesses (in parallel) verifier challenges $r_1, \dots, r_N \in \mathbb{F}$ as well as prover responses $p_1, \dots, p_N: \mathbb{F} \rightarrow \mathbb{F}$, where $N = \text{poly}(n)$, and simulates the protocol. Of course, a malicious prover would send incorrect p_i ’s and as well as adversarial r_i ’s that happen to satisfy the protocol’s constraints checks for each pair of univariates p_{i-1}, p_i .

To ensure that this cannot happen, the verifier will not just blindly guess the r_i ’s. For every $i \in [N]$ the verifier V will create a low-space Avoid instance $C_{x_i}: [d+1] \rightarrow \mathbb{F}$ whose image is the first $d+1$ roots of $f_i - p_i$, where f_i is the correct univariate at that round and d is the degree bound. Then the algorithm will run the Avoid algorithm on x_i to obtain r_i (i.e., guessing a witness for this algorithm rather than guessing r_i itself). Intuitively we are reducing the derandomization of a specific polynomial identity testing problem (for which the r_i ’s are used) to solving low-space Avoid. Indeed, if $f_i \neq p_i$, then the algorithm will output r_i that

is a non-root of $f_i - p_i$, since the $d + 1$ roots are all in the range of C_{x_i} . In this case, the simulated protocol continues to the next round with a false claim, and by induction, at the end of the simulated protocol the verifier will catch the false final claim.

The verifier V will create x_i that is the partial transcript up until that point. The main technical challenge in the proof is designing C_{x_i} that runs in low space. Recall that “low” is with respect to the index length $|x_i|$ and to the output length $\log(|\mathbb{F}|)$. The first issue can be handled easily, since computing f_i can be done in space $\text{poly}(|\varphi|)$ and V can add polynomial padding to x_i . The second issue is more subtle: we cannot have C_{x_i} enumerating over the roots of $f_i - p_i$, as that would take space $\log(|\mathbb{F}|)$; we therefore will have C_{x_i} factor $f_i - p_i$ and output the corresponding root. Thus, we need to be able to perform factoring of degree- d polynomials in space $\ll \log(|\mathbb{F}|)$, which calls for a tailor-made version of $\text{IP} = \text{PSPACE}$. Loosely speaking, this version will use an arithmetization (over a suitable field) that will allow each C_{x_i} to perform arithmetic operations and factor degree- d polynomials in space $\text{polyloglog}(|\mathbb{F}|)$. See [Section 6.1](#) for technical details.

2.2 Plasticity from average-case hardness of low-space Avoid

A “master result” from which the other main results will be derived asserts that under [Assumption 1.1](#), any problem in P can be solved on average by a low-space “Arthur-Merlin” (AM) interactive proof system that uses a non-standard feature. Specifically, the AM verifier uses time $\text{poly}(n)$ and space n^ε , sends one message to the prover, and receives a response of n^ε bits from the prover (where $\varepsilon > 0$ is arbitrarily small). The non-standard feature is that the verifier is also allowed persistent two-way read-only access to its random coins, akin to BP^*L machines described in [Section 1.2](#).

Theorem 2.1 (trading space for non-determinism and randomness; see [Theorem 5.1](#)). *Suppose that [Assumption 1.1](#) holds with $p(n) = n^{-\omega(1)}$. Then for every $L \in \text{P}$ and every constant $\gamma > 0$ there is an AM verifier V with the following properties:*

- **Complexity:** *The verifier V runs in time $\text{poly}(n)$, uses at most n^γ space, and has prover message length n^γ . It also uses $\text{poly}(n)$ bits of persistent read-only randomness, and issues oracle queries to $C = \{C_x\}$ on inputs of length $\text{poly}(n)$.¹¹*
- **Average-case completeness.** *With probability at least $1 - n^{-\omega(1)}$ over a choice of $x \in \{0, 1\}^n$ and over V ’s random coins, if $x \in L$ then there is a prover message causing V to accept.*
- **Worst-case soundness.** *For every $x \notin L$, the probability over V ’s random coins that there is a prover message causing V to accept is at most $n^{-\omega(1)}$.*

The motivating observation for proving [Theorem 2.1](#), which is well-known, is that *non-solutions to Avoid can be compressed*. To see this in our context, consider an Avoid family $\{C_x\}$, and fix some input x that will be available to all algorithms that we consider. Then, for any $y \in \text{Range}(C_x)$, we can represent y by a smaller z such that $y = C_x(z)$. In particular, if $C_x: \{0, 1\}^\ell \rightarrow \{0, 1\}^k$ has large polynomial stretch $k \gg \ell$ and is evaluable in low space, then we can represent $y \in \{0, 1\}^k$ by $z \in \{0, 1\}^\ell$ that is polynomially shorter, and there is a low-space algorithm that de-compresses z back into y (using x).

Our assumption is that any polynomial-time algorithm for Avoid fails on most inputs x , with respect to some family $\{C_x\}$ that has a large polynomial stretch and is low-space evaluable. What follows is that *the output of any polynomial-time algorithm can be compressed*, on a random x . In particular, for any $L \in \text{P}$ decided by a polynomial-time M_L , consider an algorithm that, given x , outputs the computation of $M_L(x)$. The conclusion is that *the computation of $M_L(x)$ has a short description*. We will leverage this short description into a probabilistic verifier that decides L with low space and low communication from the prover.

¹¹For readers from complexity theory, we note that the randomness used by the verifier is important, and deterministically simulating it may lead to a prohibitive increase in the amount of non-determinism. Specifically, recall that typical approaches for simulating randomness enumerate over a set of pseudorandom strings. In interactive proof systems, the prover will send a separate proof corresponding to each pseudorandom string. However, the number of pseudorandom strings may be more than the input length n (which increases the prover message length to be super-linear), and this super-linear overhead may be inherent for interactive proof systems; see [Section 7](#), and also see [\[CT22, CRT25\]](#).

First step: A trade-off for sublinear-time computation. Let $L \in \mathsf{P}$, decided by M_L . Consider an algorithm A that gets input $x \in \{0, 1\}^n$ and outputs a PCP witness π_x for $M_L(x)$ (i.e., a PCP encoding of the computation of $M_L(x)$), with respect to some PCP verifier computable in low space. Let $C_x: \{0, 1\}^\ell \rightarrow \{0, 1\}^k$ be the hard Avoid family, where $\ell \ll k \ll n$, and pretend for a moment that the PCP witness π_x for x has length k . Then, by our hardness assumption, for most x , there is a short $z \in \{0, 1\}^\ell$ such that $\pi_x = C_x(z)$.

The probabilistic verifier V_0 for L gets input x , guesses $z \in \{0, 1\}^\ell$, and runs the PCP verifier $V_{\text{PCP}}(x)$. Whenever V_{PCP} queries its proof, V_0 answers by computing $C_x(z)$. Note that the proof length is ℓ , which is far shorter than the trivial k , and that V_0 is a low-space algorithm (since C_x is evaluable in low space, and V_{PCP} is computable in low space). As for correctness, if $x \notin L$ then V_0 rejects any guess of z , with high probability; and for most x 's, if $x \in L$ then there is a guess of z such that V_0 accepts.

The idea above works when L is a problem computable in sub-linear time n^ε . (When working in the RAM model, as we do in this paper, this is a natural class of problems, since sub-linear time RAM algorithms can be thought of as a computationally bounded version of shallow decision trees.) In this case, PCP proofs are of length $k = n^{O(\varepsilon)}$, and the verifier V_0 decides L using space and communication approximately $\ell \ll n^\varepsilon$. This proof for sub-linear time is fleshed out in Theorem 5.1.

However, for a general $L \in \text{DTIME}[n^c]$, PCP witnesses will be of length at least $n^c \gg n$. The hard Avoid instances C_x that we consider have output length $k \ll n$ (and recall that this is important, as mentioned after Assumption 1.1), and so there is no hope of using them to compress PCP witnesses as above.

Second step: Lifting to a general $L \in \mathsf{P}$. We resolve this problem by decoupling the input on which we want to decide L from the input on which we leverage the hardness of Avoid. That is, to decide if $x \in L$, the verifier will use a hard Avoid instance indexed by a string different from x .

Let $L \in \text{DTIME}[n^c]$, and let $\{C_x\}$ be the hard Avoid family, stretching $\ell(n) = n^\delta$ bits to $k(n) = n^\varepsilon$ bits, where $\delta \ll \varepsilon$. Our verifier V gets input $x \in \{0, 1\}^n$, and pads it with a random y to $xy \in \{0, 1\}^N$, where $N = n^{O(c/\varepsilon)}$ is large enough so that $k(N) = N^\varepsilon = n^{O(c)}$ is the length of PCP proofs certifying that $x \in L$. With high probability, C_{xy} is a hard Avoid instance; hence, the algorithm that gets xy and outputs the $k(N)$ -bit PCP proof π_x certifying that $x \in L$ fails. In this case there is an $\ell(N)$ -bit z such that $C_{xy}(z) = \pi_x$.

This V is quite different from V_0 above. In terms of communication, it first sends y to the prover and only then receives z and verifies it. (We can assume that V sends the random choices used for V_{PCP} along with y , using the standard trick of Babai and Moran [BM88] and relying on the fact that $z \in \{0, 1\}^\ell$ is short; see the proof of Theorem 5.4 for details.) However, since V will also need y to resolve the queries of V_{PCP} to its proof $C_{xy}(z)$, it cannot “send y and forget it”; this is why we consider V that has persistent read-only two-way access to its random tape on which y is stored. Fortunately, the prover communication and the space complexity of V are still proportional to $\ell(N) = N^\delta \ll n^c$. In particular, if the hard Avoid instance has sufficient stretch (i.e., $\ell(N)$ is small enough), then the space complexity and prover communication are sub-linear in n , i.e. of the form n^γ for an arbitrarily small $\gamma > 0$.

A technical ingredient: PCPs for sub-linear time. The proof above works if we use standard PCPs for $\text{DTIME}[n^c]$ whose verifier runs in low space. However, to use the same approach for our other results (e.g., in Section 7) and for full generality, we provide a more generic solution, which may be of independent interest. Readers who are not interested can safely skip this part.

Instead of using a PCP for deciding L , we think of the verifier V in the proof above as using a PCP for deciding $L^{\text{pad}} = \{xy : x \in L\}$ (where y is of the correct length), which is in $\text{DTIME}[N^\varepsilon]$. We will design a PCP verifier for *any* problem in $\text{DTIME}[N^\varepsilon]$, rather than only for padded problems such as L^{pad} . In fact, as usual, we will design a PCP verifier for any problem in $\text{NTIME}[N^\varepsilon]$. The main requirements will be that the proof length is sub-linear, i.e. $N^{O(\varepsilon)}$, and that the verifier runs in polylogarithmic space.

The key observation is that a sublinear-time computation reads only a short (N^ε -length) excerpt y of its input. Therefore, the computation can be decomposed into two steps: 1. guess a short excerpt y of the input and verify that the computation accepts when seeing y ; 2. verify that y is consistent with the actual input x .

The first step concerns only the short excerpt y and a suitable witness. Instead of providing y directly, the PCP provides its low-degree extension $\tilde{y}$ as the implicit input to a standard PCPP (probabilistically checkable proof of proximity) [BGH⁺05]. This PCPP checks that $\tilde{y}$ is close to the encoding of an excerpt on which the computation accepts for some witness. Since the PCPP accesses $\tilde{y}$ only as an oracle and its explicit input has length $O(\log N)$, its verifier runs in $\text{polylog}(N)$ time, and hence in $\text{polylog}(N)$ space.

For the second step, enforcing consistency between x and y , reduces to performing what is called a “zero-on-subcube test” (see [Har04]) on a careful composition of the low-degree extensions $\tilde{x}$ and $\tilde{y}$ (of x and y respectively). This test requires access to the low-degree extension $\tilde{x}$ of x , but only at a few locations. The verifier can therefore compute each requested value of $\tilde{x}$ directly from x on the fly, using $\text{polylog}(N)$ space, without ever constructing or storing the full encoding $\tilde{x}$. For details see the proof of Theorem 4.1.

2.2.1 Proofs of Theorems 1.2, 1.3, and 1.4

To prove the equivalence in Theorem 1.2 we need a minor modification of the verifier V from Theorem 2.1. Specifically, we want a verifier that, on input x , prints the PCP proof π_x in its entirety (when there is a short z such that $\pi_x = C_{xy}(z)$). This can be done by enumerating over all possible random choices for V_{PCP} , and if the latter accepts $C_{xy}(z)$ with probability 1, then outputting $C_{xy}(z)$ bit-by-bit. Note that this modification does not meaningfully increase the space complexity or the prover communication: the new verifier still uses n^γ space and n^γ -sized proofs, as well as persistent read-only access to the random padding y .

In particular, using a suitable PCP system, the new verifier can print the multi-bit output of any FP algorithm using n^γ space and prover communication, and with persistent randomness (since the output is included as part of the PCP proof; see the proof of Theorem 5.12 for details). Such a verifier can be easily simulated by an $O(n^\gamma)$ -space algorithm that has persistent access to its randomness, by enumerating over all possible prover messages. And if $\text{prBP}^*\text{L} = \text{prL}$, then the latter algorithm can be derandomized, yielding a deterministic space- $O(n^\gamma)$ algorithm that succeeds on most inputs.

This establishes the first direction in Theorem 1.2. As for the other direction, assuming that low-space simulation of FP is possible, consider the following indexed Avoid family $\mathcal{C} = \{C_x\}_x$. The circuit $C_x: \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ gets as input an ℓ -bit description of a machine M , where $\ell = n^{o(1)}$, and outputs $M(x)$ if the computation runs in low space and outputs 0^n (otherwise it outputs 0^n). Note that C_x is computable in low space. The point is that for any FP algorithm A trying to solve $\text{Avoid}_{\mathcal{C}}$, since A can be simulated over a random x by some low-space B , the output of $A(x)$ will be in the range of C_x for a random x (i.e., as long as n is sufficiently large so that the description of B is of size ℓ). Hence, $\text{Avoid}_{\mathcal{C}}$ is hard on average for FP.

Proof of Theorem 1.3. Our starting point is, again, the probabilistic verifier V from Theorem 2.1. We want to simulate it by a Σ_k machine running in linear time. The idea is to use the classical result of Nepomnjascii [Nep70], which asserts that any problem computable in non-deterministic simultaneous polynomial time and sub-linear space (i.e., space $n^{1-\Omega(1)}$) can be computed in $\Sigma_k\text{TIME}[O(n)]$ for a large enough $k \in \mathbb{N}$. Thus, our goal is to simulate the probabilistic verifier in $\text{NTIME}[\text{poly}(n), n^{1-\Omega(1)}]$.

This is essentially a derandomization task, and we will use our derandomization assumption. Specifically, recall that the verifier sends a random y and randomness for V_{PCP} and expects to receive back z . As proved by Klivans and van Melkebeek [KvM02], our assumption implies that there exists an $O(\log n)$ -space computable pseudorandom generator such that on every input x , the probability that there is z convincing V to accept remains approximately the same when using pseudorandom strings instead of random ones.

The non-deterministic machine receives z 's corresponding to all pseudorandom strings, and checks if V accepts each z when using the corresponding pseudorandom string. The key point is that each pseudorandom string is computed on-the-fly in space $O(\log n)$, and thus the total space consumption of this machine is almost identical to that of V . See the proof of Theorem 5.9 for details.

Proof of Theorem 1.4. This proof uses a standard worst-case to average-case reduction based on error-correcting codes, and leverages the fact that the decoder for such codes uses low space.

In more detail, to decide $L \in \text{DTIME}[2^{c \cdot n}]$ we consider $\bar{L}$ whose truth-tables are encodings of the truth-tables of L by a suitable systematic code. Since the code is computable in time polynomial in the truth-table length 2^n , the new problem $\bar{L}$ is in $\text{DTIME}[2^{c' \cdot n}]$ for another constant $c' > c$. By Theorem 1.2 there is an algorithm deciding $\bar{L}$ in polynomially smaller space $2^{\varepsilon \cdot n}$ on average. Combining this algorithm with the decoder for the code, which runs in time (and hence space) $\text{poly}(n)$, we obtain an algorithm that decides $\bar{L}$ in the worst-case with roughly the same space complexity (and hence also decides L in the worst-case with such complexity). The same idea applies when working with a Σ_k -machine, and since the details follow standard approaches we postpone them to Section 5.2.

2.3 A non-deterministic algorithm for low-space Avoid: Proof of Theorem 1.6

The proof of Theorem 1.6 goes in the contrapositive direction: If Avoid is hard for non-deterministic polynomial time on average, then there are no sub-exponentially strong one-way functions. The proof uses the results of Liu and Pass [LP20, LP22], who showed that an algorithm that solves a certain $L^{\text{hard}} \in \text{NP}$ on average over the uniform distribution (with success $1 - n^{-\omega(1)}$) can be used to break any one-way function.

Thus, our goal is to show that if Avoid is hard for non-deterministic polynomial time on average, then any $L \in \text{NP}$ can be solved on average over the uniform distribution in sub-exponential time. In fact, under the hardness assumption, for any $L \in \text{NP}$ we will show a probabilistic verifier with space and prover communication n^γ that decides L on average (and a sub-exponential time is obtained by enumeration). Indeed, observe that this statement is simply an extension of Theorem 2.1 to the non-deterministic setting.

Recall that in the proof of Theorem 2.1, for any $L \in \text{DTIME}[N^\varepsilon]$ we considered a deterministic algorithm for Avoid that on input xy outputs a PCP proof π_{xy} certifying that its input xy is in L . For any xy on which this algorithm fails, the probabilistic verifier V could guess z such that $\pi_{xy} = C_{xy}(z)$.

In the current proof we start with $L \in \text{NTIME}[N^\varepsilon]$, and the algorithm A for Avoid non-deterministically guesses w certifying that $xy \in L$ (otherwise it rejects) and outputs a PCP encoding $\pi_{xy,w}$ of w . Indeed, since we now assume that Avoid is hard for non-deterministic algorithms, this algorithm A also fails for a random xy . However, we need to be careful about what “fails” means: our hardness assumption is that for a random xy , the algorithm A either outputs $\perp$ on all witnesses, or outputs a string in $\text{Range}(C_{xy})$ on some witness. The key point is that by the construction of A , whenever $xy \in L$ there necessarily *is* a witness such that A outputs *some string* (of the appropriate length). Hence, we deduce that for a random xy , if $xy \in L$ then A outputs a PCP witness $\pi_{xy,w}$ certifying that $xy \in L$ such that $\pi_{xy,w} = C_{xy}(z)$ for some z .

The construction of the probabilistic verifier V is then identical to that in the proof of Theorem 2.1. For details, see Theorems 5.1 and 5.4.

An algorithm with high success probability from stronger assumptions. The proof above shows that if low-space Avoid is hard for non-deterministic polynomial time on average, then we can solve any $L \in \text{NP}$ in small space. Using the same ideas described in the previous section (for the proof of Theorem 1.2), we can also extend this conclusion to apply to any $f \in \text{FNP}$.

We now want to use the same approach as in the proof of Theorem 1.6. To reach a contradiction, we need $f \in \text{FNP}$ that is hard for sub-exponential time on $1 - n^{-\omega(1)}$ of inputs. Indeed, inverting a one-way permutation yields such hardness (other cryptographic assumptions would also suffice; see, e.g., [MR07, Corollary 5.13]), and in fact in the special case of one-way permutations (which yield a hard problem with unique solutions) the proof simplifies further. For details see of Theorem 6.11.

2.4 Hardness of Avoid for tiny-error non-deterministic algorithms: Proof of Theorem 1.8

The techniques in the proof of this result are unrelated to the techniques in the rest of the paper. Our starting point will be the proof of Ilango [Ila25] that Avoid is hard for zero-error non-deterministic algorithms, assuming that there are no NP-constructive natural properties against P/poly. We will think of this assumption as asserting that there is a polynomial-time computable generator g that fools non-deterministic algorithms.¹²

A different proof of the original result. Assuming towards contradiction that there is a zero-error non-deterministic machine A solving Avoid on average, we construct an adversary D breaking the security of the generator g . For simplicity, think of A for now as solving Avoid correctly on all inputs.

Intuitively, we want $D(y)$ to accept iff y is **not** in the pseudorandom set $\{g(x)\}_x$ (so that D accepts most strings but rejects all pseudorandom strings). However, a naive implementation of such D yields a co-nondeterministic machine, whereas we want a non-deterministic one. The idea is to use an Avoid algorithm as a certificate of not being in $\{g(x)\}$; that is, given y we construct C whose image is $\{g(x)\}$ (i.e., just use $C(x) = g(x)$), and if $y = A(C)$ then we are certain that $y \notin \{g(x)\}$. The idea extends to a non-deterministic A in the straightforward way.¹³

Of course, the trouble is that $A(C)$ outputs just one string, so it “certifies” only a single y (for not being in $\{g(x)\}$), whereas we want many y ’s to be “certified” (i.e., to be the output of A) in order to break the demi-bits generator. We will therefore allow another degree of freedom: we will have D accept iff there is a tuple $\mathbf{s} = (s_1, \dots, s_t)$ and $i \in [t]$ such that $y \oplus s_i$ is **not** in the set $\{g(x) \oplus s_i\}_x$ (for now think of $\mathbf{s}, i$ as a cumbersome way to specify a string s_i). Indeed, the way D will efficiently certify this is to non-deterministically guess $\mathbf{s}$ and check if for some $i \in [t]$ the algorithm A outputs $y \oplus s_i$ on a circuit $C_{\mathbf{s}}$ whose output-set is $\{g(x) \oplus s_i\}_{x,i}$.

If we are lucky, for each different $\mathbf{s}$ the output of $A(C_{\mathbf{s}})$ will be different, in which case all of y ’s will be “covered” (i.e., for every y there is $\mathbf{s}$ such that $y \oplus s_i$ is the output of $A(C_{\mathbf{s}})$). However, the Avoid algorithm A is hypothesized adversarially, and $\{A(C_{\mathbf{s}}) \oplus s_i\}_{\mathbf{s},i}$ might “concentrate” on a small set.

The main technical lemma, which is a generalization of the technical idea in [Ila25], asserts that this is impossible: thinking of A as an adversary that tries to “shift” each tuple $\mathbf{s}$ by its output $A(C_{\mathbf{s}})$, many y ’s will be covered.

Lemma 2.2 (covering lemma; informal, see Lemma 6.4). *Consider any adversary A that gets $\mathbf{s} = (s_1, \dots, s_t) \in \{0,1\}^{t \times n}$ and outputs $A(\mathbf{s})$. For every $\mathbf{s}$, let $A(\mathbf{s}) \oplus \mathbf{s} = (A(\mathbf{s}) \oplus s_1, \dots, A(\mathbf{s}) \oplus s_t)$ be the A -shifted tuple corresponding to $\mathbf{s}$. Then, the set of n -bit strings covered by A -shifted tuples is of size at least $2^{n-n/t}$.*

The full proof implements the idea above also in the case where A succeeds only on a fraction of inputs $C_{\mathbf{s}}$, rather than on every $C_{\mathbf{s}}$ (see the proof of Theorem 6.5 in Section 6.2.1).

Extension to tiny-error algorithms. The proof above and the generalized covering lemma already do much of the heavy lifting to support the extension to tiny-error non-deterministic algorithms.

Specifically, the proof above shows that a large fraction of strings are covered by (at least one of the) tuples; and none of the strings in the image of g are covered by any tuple, due to the zero-error property of the assumed algorithm A . This NP-property (i.e., “is there a covering tuple”) allowed us to distinguish between the set of pseudorandom strings and a large set of truly random strings.

However, once we relax A and allow it to err, the soundness property that none of the strings in the image of g are covered is no longer true. Indeed, if A makes error on κ fraction of choices of $\mathbf{s}$, then a pseudorandom string could potentially be covered by $\kappa \cdot 2^{tn}$ tuples (i.e. tuples $\mathbf{s}$ that A errs on).

Nonetheless, if κ is sufficiently small, one can hope to show that many random strings are covered by $2\kappa \cdot 2^{tn}$ tuples, and this sets them apart from pseudorandom strings. Indeed, this is precisely what we show, by extending the analysis in the proof of the covering lemma. And at that point one can apply the

¹²That is, there is no non-deterministic algorithm that accepts most strings but rejects all pseudorandom strings.

¹³That is, accept y if and only if there is w such that on input C and witness w the algorithm A outputs y .

Goldwasser-Sipser Set Size Lower Bound protocol [GS86] to distinguish these two cases (i.e., at most κ fraction of tuples vs at least 2κ fraction of tuples). This is why our assumption is stronger than in [Ila25]: we require the generator to be secure against AM adversaries (implementing the set lower bound protocol).

2.5 A targeted hitting-set generator for a subclass of coNP

Finally, we describe the construction of a non-deterministic targeted hitting-set generator for coNP procedures that use few non-deterministic bits. Recall that we assume sub-exponentially secure one-way functions. We will again use the results of Liu and Pass [LP20, LP22], and this time we will also rely on the fact that the average-case hard language $L^{\text{hard}} \in \text{NP}$ in their result is actually in *non-deterministic sub-linear time*, i.e. $L^{\text{hard}} \in \text{NTIME}[n^\varepsilon]$ for a suitable small $\varepsilon > 0$.¹⁴

Recall that in the proof of Theorem 1.6, we assumed towards a contradiction that a non-deterministic algorithm $A(x)$ outputting a PCP witness π_x for L^{hard} failed to solve Avoid for a random xy , and deduced that we could compress π_x (to z such that $C_{xy}(z) = \pi_x$) for a random x . In the current proof, we will assume towards a contradiction that any non-deterministic generator $H(x)$ fails to output a hitting-set for some coNP distinguisher $D(x, \cdot)$ (i.e., it either prints $\perp$ on all witnesses, or outputs strings that $D(x, \cdot)$ rejects), and we will deduce a corresponding space savings for computing L^{hard} .

The main technical tool underlying the proof is a new reconstructive targeted hitting-set generator that is based on the hardness of problems in sub-linear time. Specifically, for any $L \in \text{NTIME}[n^\varepsilon]$, we show a construction of $H = H_L$ that outputs $n^{O(\varepsilon)}$ pseudorandom strings of length (say) $n^{\varepsilon/2}$, and satisfies the following: if $H(x)$ fails to hit $D(x, \cdot)$ (that uses few non-deterministic bits), then another algorithm $R(x)$ decides $L(x)$ in space n^δ , for an arbitrarily small $\delta \ll \varepsilon$. In particular, using $L = L^{\text{hard}}$, the reduction of [LP20, LP22] will allow us to invert any one-way function in time 2^{n^δ} .

The new targeted hitting-set generator. The high-level idea behind our construction of $H = H_L$ and R comes from the work of van Melkebeek and Sdroieviski [vMS23a]. The generator $H(x)$ guesses an NP-witness $w \in \{0, 1\}^{n^\varepsilon}$ for $L(x)$, and if w is convincing then the generator transforms w into a PCP witness $\pi = \pi_w$ of length $|\pi| = n^{O(\varepsilon)}$ for the computation of $L(x)$. Then it applies a classical pseudorandom generator – for simplicity, think of the Nisan-Wigderson [NW94] generator – with π as the hard truth-table. The number of outputs of $\text{NW}(\pi)$ is indeed $\text{poly}(|\pi|) = n^{O(\varepsilon)}$ as we wanted.

The properties of the NW generator guarantee that if D distinguishes $\text{NW}(\pi)$ from random strings, then there exists a circuit C of size n^δ (where $\delta > 0$ can be arbitrarily small¹⁵) such that $C^D(i) = \pi_i$ for all $i \in [|\pi|]$. The reconstruction R gets input x , guesses C of size n^δ , and runs a PCP verifier for L , resolving the PCP verifier’s queries to bits i of the proof by simulating $C^D(i)$. For $R(x)$ to run in low space, we need:

1. An $n^{O(\varepsilon)}$ -size PCP with a low-space verifier.
2. The simulation of $C^D(i)$ to run in low space.

For the first item above, we use the PCP verifier described in Section 2.2. For the second item, we use the fact that D needs only few non-deterministic bits, and we replace the NW generator by a more suitable generator that needs a more relaxed access to D rather than oracle queries (namely, the generator of Miltersen and Vinodchandran [MV05]). For full details, see the proof of Theorem 7.7.

3 Preliminaries

The machine model throughout the paper is the RAM model (this is important, as we will be studying sublinear-time computation). We note that our simulations in Theorems 1.2, 1.3, 1.4 and 2.1 do not rely on the RAM model, and can be implemented on a multitape Turing machine.

¹⁴Recall that we work with RAM machines in this paper. Also, for non-deterministic machines, the notion of sub-linear time is even more natural: many standard NP-complete problems are computable in sub-linear time (e.g., clique), and every NP-complete problem can be padded to another NP-complete problem that is computable in sub-linear time.

¹⁵We pay for a small δ by a larger polynomial blow-up $N^{O(\varepsilon)}$ in the number of output strings of $H(xy)$, which we can afford.

We call a function $f : \mathbb{N} \rightarrow \mathbb{N}$ nice if there is a deterministic algorithm that given n as input, outputs $f(n)$ in time $\text{polylog}(n, f(n))$.

3.1 Indexed circuit families

We consider indexed families of circuits. That is, every string $x \in \{0, 1\}^*$ is associated with some circuit C_x , and the circuit family is $\{C_x\}_{x \in \{0, 1\}^*}$.

Definition 3.1 (indexed circuit families). *Let $\mathcal{C} = \{C_x\}_{x \in \{0, 1\}^*}$ be an indexed family of circuits. We say that $\mathcal{C} \in \mathcal{C}[\ell_{in} \rightarrow \ell_{out}, s]$ if each $C_x \in \mathcal{C}$ has size $s(|x|)$ and $\ell_{in}(|x|)$ bits of input and $\ell_{out}(|x|)$ bits of output.*

For convenience, we will frequently abuse notation by saying “Let $\mathcal{C}[\ell_{in} \rightarrow \ell_{out}, s]$ be an indexed family of circuits”, rather than saying “Let $\mathcal{C} \in \mathcal{C}[\ell_{in} \rightarrow \ell_{out}, s]$ be an indexed family of circuits”.

We will be interested in indexed circuit families of different levels of uniformity.

Definition 3.2 (P-evaluable). *Let $\ell_{in} < \ell_{out} < s : \mathbb{N} \rightarrow \mathbb{N}$ be nice functions, and let $\mathcal{C}[\ell_{in} \rightarrow \ell_{out}, s] = \{C_x\}_{x \in \{0, 1\}^*}$ be an indexed family of circuits. We say that $\mathcal{C}$ is P-evaluable if there is a Turing machine M that gets input $(x, y, i) \in \{0, 1\}^n \times \{0, 1\}^{\ell_{in}(n)} \times [\ell_{out}(n)]$, runs in time $\text{poly}(n)$, and prints the i -th output bit of $C_x(y)$.*

Definition 3.3 (SPACE-evaluable). *Let $\ell_{in} < \ell_{out} < s : \mathbb{N} \rightarrow \mathbb{N}$ be nice functions, and let $\mathcal{C}[\ell_{in} \rightarrow \ell_{out}, s] = \{C_x\}_{x \in \{0, 1\}^*}$ be an indexed family of circuits. We say that $\mathcal{C}$ is SPACE($S(n)$)-evaluable if there is a Turing machine M that gets input $(x, y, i) \in \{0, 1\}^n \times \{0, 1\}^{\ell_{in}(n)} \times [\ell_{out}(n)]$, runs in space $S(n)$, and prints the i -th output bit of $C_x(y)$.*

3.2 Complexity-theoretic notions, classes, and results

We recall some standard definitions that will be used throughout the paper.

Definition 3.4 (samplable distributions). *A distribution ensemble $\{D_n\}_{n \in \mathbb{N}}$ is called samplable if there exists a probabilistic polynomial-time algorithm A such that for every $n \in \mathbb{N}$, $A(1^n) \sim D_n$. That is, A outputs a sample distributed exactly according to D_n .*

Definition 3.5 (search NP with few non-deterministic bits). *Let $\mathcal{R} \subseteq \{0, 1\}^* \times \{0, 1\}^*$ be a total search problem and let $\mathcal{D} = \{D_n\}$ be an ensemble of samplable distributions. We say that $(\mathcal{R}, \mathcal{D})$ is in average-case SearchNTimeGuess $[T(n), g(n)]$ with success p if there exists a non-deterministic Turing machine M such that for every input length n , using $g(n)$ -bits of non-determinism and running in time $T(n)$, with probability at least p over the choice of input x drawn from the input distribution D_n :*

- On all accepting paths, $M(x)$ outputs some y such that $(x, y) \in \mathcal{R}$.
- $M(x)$ has at least one accepting path.

Note that the definition above allows M to err on the “bad” inputs x , i.e. $M(x)$ might have accepting paths that output y where $(x, y) \notin \mathcal{R}$.

Definition 3.6 (low space with persistent randomness). *Let $s : \mathbb{N} \rightarrow \mathbb{N}$ be any function. A Turing Machine M is called a BP*SPACE $[s(n)]$ machine if M is a probabilistic Turing Machine that on an n -bit input x , uses $s(n)$ memory bits, runs for at most $2^{O(s(n))}$ time and has 2-way access to its randomness.*

Co-nondeterministic procedures. For a co-nondeterministic procedure P taking input x and (co-)nondeterministic choice w , we will often suppress w in our notation, and denote $P(x)$; the meaning is then as follows:

Definition 3.7. For a co-nondeterministic procedure P taking input x and witness w , we denote $P(x) = 0$ if there is w such that $P(x)$ rejects with the (co-)nondeterministic choice w , and $P(x) = 1$ otherwise. We will sometimes consider co-nondeterministic procedures that take input (x, r) rather than only x , in which case the notation $P(x, r)$ is defined identically.

We will often think of $P(x, r)$ as a probabilistic procedure where x is the input and r is the randomness, and the co-nondeterministic choice is made after (x, r) are fixed. This is the case for Arthur-Merlin verifiers.

Three standard results. We recall the classical result of Nepomnjascii simulating low-space non-deterministic time in linear time with alternations, the set lower bound protocol of Goldwasser and Sipser, and the round reduction result of Babai and Moran.

Theorem 3.8 ([Nep70]). For every positive constant d and ε , there exists an integer k such that $\text{NTISP}[n^d, n^{1-\varepsilon}] \in \Sigma_k \text{TIME}[O(n)]$.

Theorem 3.9 (Goldwasser–Sipser set-size lower bound [GS86]). Let

$$S_x = \left\{ y \in \{0, 1\}^{m(|x|)} : \exists z R(y, z) = 1 \right\},$$

where R is polynomial-time computable and m is polynomially bounded. Then there is a two-message AM protocol such that, on input x and an integer threshold τ ,

- if $|S_x| \geq 2\tau$, then Arthur accepts with probability at least $2/3$ for some honest Merlin strategy;
- if $|S_x| \leq \tau$, then Arthur accepts with probability at most $1/3$ for every (malicious) Merlin strategy.

We recall the round reduction result for AM from [BM88], tailored for our needs.

Lemma 3.10. Let (V_1, P_1, V_2, P_2) be a 2-round Arthur-Merlin protocol deciding a promise problem (Π_Y, Π_N) with completeness $1 - c$, constant soundness $s \leq 1/(2\sqrt{2})$ and message complexity v_1, p_1, v_2, p_2 . Then, for a suitable $N_1 = \text{polylog}(n)$, there is a 1-round Arthur-Merlin protocol (V', P') that decides (Π_Y, Π_N) with completeness at least $1 - 2N_1 p_1 c$, soundness $2^{-\text{polylog}(n)}$ and message complexity $v' = \text{polylog}(n)(v_1 + p_1 v_2)$, $p' = \text{polylog}(n)(p_1 + p_1 p_2)$.

Proof. On input x , the new verifier V' works as follows. Let $N_1 = \text{polylog}(n)$ and $N_2 = 2p_1$. For each $i \in [N_1]$ and $j \in [N_2]$, the verifier V' samples $\alpha_i \in \{0, 1\}^{v_1}$ and $\beta_{i,j} \in \{0, 1\}^{v_2}$ and sends them to P' . The honest prover P' simulates $N_1 \cdot N_2$ copies of the original protocol by taking α_i as the first message by V_1 and $\beta_{i,j}$ as the second message by V_2 , and replies with u_i and $w_{i,j}$ as message by P_1 and P_2 respectively. The verifier V' accepts if and only if all $N_1 \cdot N_2$ copies of simulations $V_2(x, \alpha_i, u_i, \beta_{i,j}, w_{i,j})$ accepts.

Completeness. Suppose that $x \in \Pi_Y$, and let P_1, P_2 be honest prover strategies for the original protocol. The honest prover P' uses these strategies in each simulated copy. Each copy has the same marginal distribution as an execution of the original protocol with the honest provers, and thus rejects with probability at most c . By a union bound over the $N_1 N_2$ simulated copies, the probability that V' rejects is at most

$$N_1 N_2 c = 2N_1 p_1 c.$$

Soundness. Suppose that the input $x \in \Pi_N$. For any fixed α, u , denote

$$q_{\alpha, u} = \Pr_{\beta \in \{0, 1\}^{v_2}} [\exists w, V_2(x, \alpha, u, \beta, w) = 1] \quad \text{and} \quad q_\alpha = \max_u q_{\alpha, u}.$$

By the soundness of the original protocol, we have $\mathbb{E}_\alpha [q_\alpha] \leq s$.

For each $i \in [N_1]$, the N_2 tests $V_2(x, \alpha_i, u_i, \beta_{i,j}, w_{i,j})$ pass if and only if there is a fixed u_i such that all $\beta_{i,j}$ would pass the test. For any fixed u_i , this happens with probability at most $q_{\alpha_i, u_i}^{N_2} \leq q_{\alpha_i}^{N_2}$ over $\beta_{i,1}, \dots, \beta_{i,N_2}$. By a union bound over 2^{p_1} possible choices of u_i , with probability at most $\min(1, 2^{p_1} \cdot q_{\alpha_i}^{N_2})$ over $\beta_{i,1}, \dots, \beta_{i,N_2}$ there is a u_i such that these N_2 tests pass. And the probability of accepting over randomly chosen α_i is

$$\mathbb{E}_{\alpha}[\min(1, 2^{p_1} \cdot q_{\alpha}^{N_2})] \leq 2^{p_1/N_2} \cdot \mathbb{E}[q_{\alpha}] \leq \sqrt{2}s.$$

Finally, each of these N_1 simulations is independent, and the overall soundness is upper bounded by $(\sqrt{2}s)^{\text{polylog}(n)} = 2^{-\text{polylog}(n)}$.

Message Complexity. Message complexity follows from the description of the protocol. $\square$

3.3 Range Avoidance

For a circuit $C: \{0,1\}^{\alpha} \rightarrow \{0,1\}^n$, recall that $\text{Range}(C) = \{y: \exists x, C(x) = y\}$.

Definition 3.11 (range avoidance). *In the Range Avoidance problem (Avoid) the input is $C: \{0,1\}^{\alpha} \rightarrow \{0,1\}^n$ where $\alpha < n$ and the goal is to find an n -bit $y \notin \text{Range}(C)$. We refer to $\alpha(n) \mapsto n$ as the stretch of the instance C .*

We will also consider a variant of range avoidance in which the input is not a circuit, but rather an input x that defines a circuit C_x . In the paper we will be considering cases in which the mapping $x \mapsto C_x$ and vice versa are efficiently computable.

Definition 3.12 (range avoidance on constructible circuits). *Let $\mathcal{C}[\ell_{in} \rightarrow \ell_{out}, s] = \{C_x\}_{x \in \{0,1\}^s}$ be an indexed family of circuits. In the Range Avoidance for $\mathcal{C}$ problem ($\text{Avoid}_{\mathcal{C}}$) the input is $x \in \{0,1\}^n$, and the goal is to find $y \in \{0,1\}^{\ell_{out}(n)}$ such that $y \notin \text{Range}(C_x)$.*

Note that if the mapping $C_x \mapsto x$ is efficiently computable, then average-case hardness of Avoid on a distribution of circuits C_x when $x \sim \mathcal{D}$ implies average-case hardness of $\text{Avoid}_{\mathcal{C}}$ on the distribution $x \sim \mathcal{D}$.

3.4 Our conclusions imply that $\text{P} \neq \text{PSPACE}$

We show that the average-case conclusions in our main results imply that $\text{P} \neq \text{PSPACE}$. The argument is essentially the same as the one used by Williams [Wil25] to show that $\text{P} \subseteq \cap_{\epsilon > 0} \text{SPACE}[n^{\epsilon}]$ would separate P from PSPACE . The only difference is that our assumption (of low-space simulation) holds only on average, so we will carry the argument relying on a stronger space hierarchy theorem.

Theorem 3.13. *For every $C > c$ there is a problem $L \in \text{SPACE}[n^C]$ that is hard for $\text{SPACE}[n^c]$ infinitely often on all inputs; that is, for every n^c -space machine M there are infinitely many $n \in \mathbb{N}$ such that $M(x) \neq L(x)$ for all $x \in \{0,1\}^n$.*

Proof sketch. This is essentially the standard proof of the space-hierarchy theorem, with the additional observation that diagonalizing against a machine M can be done on all inputs of any length $n \in \mathbb{N}$ that is associated with M , rather than only to one input of length n .¹⁶ $\square$

Theorem 3.14. *Assume that for some fixed $c > 0$, for every $L \in \text{P}$ there is a space- n^c machine M_L such that for every $n \in \mathbb{N}$ there is $x \in \{0,1\}^n$ such that $M_L(x) = L(x)$. Then $\text{P} \neq \text{PSPACE}$.*

Proof. Let $L \in \text{SPACE}[n^{2c}]$ that is hard for $\text{SPACE}[n^c]$ infinitely often on all inputs (as in Theorem 3.13). Assuming that $\text{P} = \text{PSPACE}$, we deduce $L \in \text{P}$. But then by our assumption there is M_L using space n^c that agrees with L on some input of every length, a contradiction. $\square$

¹⁶In more detail, associate every machine with infinitely many input lengths. On input any $x \in \{0,1\}^n$ associated with M , the machine for the hard problem simulates M (with its computation truncated to use at most n^c space) and outputs a different value than $M(x)$.

4 PCP for Sublinear Time

In this section we construct a PCP for $\text{NTIME}[n^\varepsilon]$ that has proof size $n^{O(\varepsilon)}$ and a verifier running in space $\text{polylog}(n)$ (and making $\text{polylog}(n)$ non-adaptive queries).

Theorem 4.1 (a PCP for non-deterministic sublinear time). *Let $\varepsilon \in (0, 1)$ be a constant and $s \geq 2^{-\text{polylog}(n)}$, and let $L \in \text{NTIME}[n^\varepsilon]$. There is a probabilistic verifier V that gets input $x \in \{0, 1\}^n$ and oracle access to a proof π of size $n^{O(\varepsilon)}$ with the following properties.*

- **Completeness:** For $x \in L$, $\exists \pi$ such that

$$\Pr[V^\pi(x) = 1] = 1.$$

- **Soundness:** For $x \notin L$, $\forall \pi'$

$$\Pr[V^{\pi'}(x) = 0] \geq 1 - s.$$

- **Complexity:** The verifier V runs in time $\text{poly}(n)$ and space $\text{polylog}(n)$, and non-adaptively queries $\text{polylog}(n)$ positions in π .

Moreover, the accepting proof π in completeness can be deterministically computed in time $\text{poly}(n)$, from x and the n^ε -size witness witnessing $L \in \text{NTIME}[n^\varepsilon]$.

4.1 Known technical tools

For the construction we will need a probabilistically checkable proof of proximity, and a “zero-on-subcube test”, similar to the one constructed by Harsha [Har04]. For concreteness, we will use a test based on the PCPP of Ben-Sasson *et al.* [BGH⁺05].

Theorem 4.2 (the PCPP of Ben-Sasson *et al.* [BGH⁺05, Theorem 2.5]). *Suppose that L is a pair language, let n denote the length of the explicit input, K denote the length of the implicit input and set $m \triangleq n + K$. If $L \in \text{NTIME}[T(m)]$ ¹⁷ for some non-decreasing function $T : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$, then for every constant $s > 0$, we have $L \in \text{PCPP}_{s,\delta}[r, q, t]$, for*

1. Proximity parameter $\delta(m) = 1/\text{polylog } T(m)$,
2. Randomness complexity $r(m) = \log T(m) + O(\log \log T(m))$,
3. Query complexity $q(m) = \text{polylog } T(m)$,
4. Verification time $t(n, K) = \text{poly}(n, \log K, \log T(m))$.

Moreover, the PCPP verifier makes only non-adaptive queries; and for $(x, y) \in L$, an accepting PCPP proof π can be deterministically computed in time $\text{poly}(T(m))$ from (x, y) and from a $T(m)$ -length witness for $(x, y) \in L$.¹⁸

Since we are willing to afford a multiplicative $\text{polylog}(T(m))$ increase to the query complexity, we can improve the soundness to $2^{-\text{polylog}(T(m))}$ in the above theorem by performing $\text{polylog}(T(m))$ repetitions on the same proof oracle.

¹⁷The notation NTIME in this result refers to decidability by multitape Turing machines (and hence the main application is for $T(m)$ that is at least linear), whereas in this paper we work in the RAM model. Whenever using Theorem 4.3 we will address the machine model explicitly.

¹⁸In [BGH⁺05] it is not explicitly stated that a PCPP proof can be efficiently computed from an NP witness, but this fact is frequently cited as implicit in their work (see, e.g., [ANOS24, Theorem 2.18], [CKLR11, Theorem 50 and the subsequent discussion], [CT22, Theorem 3.17], [BR22, Theorem 2.8], and [BBH⁺19, Theorem 2.5], some of which omit features of the original result).

Corollary 4.3. Suppose that L is a pair language, let n denote the length of the explicit input, K denote the length of the implicit input and set $m \triangleq n + K$. If $L \in \text{NTIME}[T(m)]$ for some non-decreasing function $T : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$. Then for any $s \geq 2^{-\text{polylog}(T(m))}$, we have $L \in \text{PCPP}_{s,\delta}[r, q, t]$, for

1. Proximity parameter $\delta(m) = 1/\text{polylog } T(m)$,
2. Randomness complexity $r(m) = \text{polylog } T(m)$,
3. Query complexity $q(m) = \text{polylog } T(m)$,
4. Verification time $t(n, K) = \text{poly}(n, \log K, \log T(m))$.

Moreover, the PCPP verifier makes only non-adaptive queries; and for $(x, y) \in L$, an accepting PCPP proof π can be deterministically computed in time $\text{poly}(T(m))$ from (x, y) and from a $T(m)$ -length witness for $(x, y) \in L$.

Definition 4.4. Fix a finite field $\mathbb{F}$, positive integers m and d and set $H \subseteq \mathbb{F}$. A degree d polynomial $f : \mathbb{F}^m \rightarrow \mathbb{F}$ is Zero on the Subcube H^m if its restriction to H^m is identically 0. We denote the set of all degree- d polynomials that are Zero on the Subcube H^m by $\text{ZoS}(\mathbb{F}, m, d, H)$.

Theorem 4.5 (a PCPP for a zero-on-subcube test). Let L be the pair language $\{((|\mathbb{F}|, m, d, |H|), f) : f \in \text{ZoS}(\mathbb{F}, m, d, H)\}$ where f is represented by a truth-table, the set H is the first¹⁹ $|H|$ elements of $\mathbb{F}$, and the integers $|\mathbb{F}|, m, d, |H|$ are represented in binary. Then for any $s \geq 2^{-\text{poly}(m, \log |\mathbb{F}|)}$, we have $L \in \text{PCPP}_{s,\delta}[r, q, t]$, for

1. Proximity parameter $\delta = 1/\text{poly}(m, \log |\mathbb{F}|)$,
2. Randomness complexity $r = \text{poly}(m, \log |\mathbb{F}|)$,
3. Query complexity $q = \text{poly}(m, \log |\mathbb{F}|)$,
4. Verification time $t = \text{poly}(m, \log |\mathbb{F}|)$.

Moreover, the PCPP verifier makes only non-adaptive queries, and the accepting proof π in completeness can be deterministically computed in time $\text{poly}(|\mathbb{F}|^m)$ from the implicit input f .

Proof. The pair language L has a first input of length $n = \log(|\mathbb{F}|) + \log(m) + \log(d) + \log(|H|)$ and a second input of length $K = |\mathbb{F}|^m \cdot \log(|\mathbb{F}|)$. It can be decided by a multitape Turing machine (by brute-force enumeration over $\mathbb{F}^m$) in time $T(n + K) = \text{poly}(|\mathbb{F}|^m) = \text{poly}(2^n, K)$. Using Theorem 4.3 yields a PCPP with the stated parameters. $\square$

4.2 Proof of Theorem 4.1

Let $L \in \text{NTIME}[n^\varepsilon]$, i.e. there exists a deterministic verifier V_0 such that $x \in L \cap \{0, 1\}^n \iff \exists w, V_0^x(\alpha, w) = 1$ and V_0 runs in time $O(n^\varepsilon)$. We denote by $\alpha = \alpha(n)$ the binary representation of n where $|\alpha| = \lceil \log n \rceil$.

Arithmetic setting. Consider the following arithmetization of an input x . Set $h \triangleq n^{\varepsilon'}$ for some $\varepsilon' < \varepsilon$. Let $Q \in [h^3, 2h^3]$ be a power of 2 and let $\mathbb{F} \triangleq \mathbb{F}_Q$ be a field. In particular, each field element can be encoded using $3 \log(n)$ bits. Let $H = [h] \subset \mathbb{F}$. Let $m_1 \geq m_2$ be the smallest integers such that $h^{m_1} \geq n$ and $h^{m_2} \geq n^\varepsilon$. We interpret x as a function $x : H^{m_1} \rightarrow \{0, 1\}$ via some canonical mapping $g : H^{m_1} \rightarrow [n]$ and let $\tilde{x} : \mathbb{F}^{m_1} \rightarrow \mathbb{F}$ be the unique extension of x of individual degree $(h - 1)$.

We now define polynomials that are essentially arithmetized “excerpts” of x . Specifically, consider any $y : H^{m_2} \rightarrow H^{m_1} \times \{0, 1\}$, whose truth-table is a string of size $O(m_1 \cdot \log(n) \cdot n^\varepsilon)$. We parse such y as a table of n^ε tuples $(c_1, \dots, c_{m_1}, b)$, interpreted as “ $x(c_1, \dots, c_{m_1}) = b$ ”; that is, the $g(c_1, \dots, c_{m_1})$ -th bit of x is b .

¹⁹Any reasonable ordering of the elements of $\mathbb{F}$ suffices for this result (i.e., as long as the set H can be enumerated in time $\text{poly}(|\mathbb{F}|)$). We will instantiate the result with finite field of characteristic 2, in which case H is the set of the first h field elements in the lexicographic ordering of their binary representations.

We then arithmetize such y with $m_1 + 1$ polynomials $\tilde{y}_i : \mathbb{F}^{m_2} \rightarrow \mathbb{F}$, where each $\tilde{y}_i$ is the unique extension of the i -th output coordinate of y of individual degree $(h - 1)$. For ease of presentation, we use $\tilde{y}$ to denote $\{\tilde{y}_1, \dots, \tilde{y}_{m_1+1}\}$. Note that the truth-table of $\tilde{y}$ has size $(m_1 + 1) \cdot |\mathbb{F}|^{m_2} \cdot \log(n) = n^{O(\varepsilon)}$.

Next, we say that y agrees with x if for every tuple $(c_1, \dots, c_{m_1}, b)$ in the truth-table of y it holds that $x(c_1, \dots, c_{m_1}) = b$. Notice that y agrees with x if and only if the polynomial $\tilde{p} : \mathbb{F}^{m_2} \rightarrow \mathbb{F}$ defined as

$$\tilde{p}(\cdot) \triangleq \tilde{x}(\tilde{y}_1(\cdot), \dots, \tilde{y}_{m_1}(\cdot)) - \tilde{y}_{m_1+1}(\cdot) \quad (4.1)$$

is all zero over the subcube H^{m_2} .

An alternative verifier for L . At a high level, we will be interested in y that is the excerpt of x relevant to verification of $V_0^x(\alpha, w)$, for a given w . We will think of the verification $V_0^x(\alpha, w)$ as split into two parts:

1. Whether V_0 accepts seeing the excerpt y ;
2. Whether y agrees with x .

In more detail, we start by defining a deterministic verifier V' as follows: $V'(\alpha, \tilde{y}, w)$ first parses $\tilde{y}$ as truthables of $m_1 + 1$ polynomials $\tilde{y}_i : \mathbb{F}^{m_2} \rightarrow \mathbb{F}$, reads off $y : H^{m_2} \rightarrow H^{m_1} \times \{0, 1\}$ directly from these polynomials. It rejects if y takes undefined value at any position. Next, it verifies that for each $i \in [m_1 + 1]$, $\tilde{y}_i$ is the unique low degree extension of y_i . This is done by interpolating the polynomial using values of y_i , and checking against every single evaluation in $\tilde{y}_i$. This can be done by brute force in $O(m_1 \cdot |\mathbb{F}|^{m_2} \cdot \log n) = n^{O(\varepsilon)}$ time.

Next, it simulates $V_0^x(\alpha, w)$ in the following manner: Whenever V_0 is about to query x_i , the verifier V' reads the next tuple $(c_1, \dots, c_{m_1}, b)$ from y , immediately rejects if $i \neq g(c_1, \dots, c_{m_1})$, and otherwise continues the simulation assuming that $x_i = b$. It then accepts iff V_0 accepts. Note that V' overall runs in polynomial time $T(|\alpha, \tilde{y}, w|) = n^{O(\varepsilon)}$.

Define the pair language L' as $(\alpha, \tilde{y}) \in L' \iff \exists w, V'(\alpha, \tilde{y}, w) = 1$. Note that $L' \in \text{NTIME}[T]$. Our construction will use two randomized verifiers:

- We invoke the PCPP from Theorem 4.3 and obtain a randomized verifier $R_1^{\tilde{y}, \pi}(\alpha)$ for L' with proximity parameter $1/\text{polylog}(n)$, randomness complexity $O(\varepsilon \cdot \log n)$, verification time (and query complexity) $\text{polylog}(n)$, and soundness error $s/2$.
- Let $R_2^{\tilde{p}, \pi}$ be the randomized verifier from Theorem 4.5 (i.e., for the all zero test) instantiated with soundness error $s/2$. Given input $x \in \{0, 1\}^n$, we will instantiate R_2 on inputs of the form $(|\mathbb{F}|, m_2, d, h)$, so that it has proximity parameter $1/\text{polylog}(n)$, randomness complexity $O(\varepsilon \cdot \log n)$, and verification time (and query complexity) $\text{polylog}(n)$.

Our final PCP verifier V for L works as follows: Given $x \in \{0, 1\}^n$, the proof consists of $\tilde{y}$ and π_1, π_2 . The randomized verifier runs the following tests and accepts if and only if both tests accept:

1. Test if $R_1^{\tilde{y}, \pi_1}(\alpha) = 1$;
2. Test if $R_2^{\tilde{p}, \pi_2}(|\mathbb{F}|, m_2, d, h) = 1$.

Complexity. The complexity of V follows from the complexities of R_1 and R_2 , both of which run in time $\text{polylog}(n)$ (hence also space $\text{polylog}(n)$, and they make $\text{polylog}(n)$ queries).

The non-trivial challenge is simulating the query access of R_2 to $\tilde{p}$. To do this, on each query $q \in \mathbb{F}^{m_2}$ the PCP verifier queries $\tilde{y}_1(q), \dots, \tilde{y}_{m_1+1}(q)$ and then computes $\tilde{x}$ on the outcomes, using the input x . Since $\tilde{x}$ is a standard low-degree extension of x , given $v \in \mathbb{F}^{m_1}$ we can compute $\tilde{x}(v)$ in time $\text{poly}(n)$ and space $\text{polylog}(n)$. Therefore, V can evaluate $\tilde{p}$ on the fly for each query, using only $\text{polylog}(n)$ space overall.

Completeness. For completeness, let $x \in L$ and let w be a witness such that $V_0^x(\alpha, w) = 1$. Let y be the correct excerpt of x from simulating $V_0^x(\alpha, w)$. Define $\tilde{y}$ as above, and the perfect completeness of R_1 and R_2 allows us to conclude the perfect completeness of V .

Soundness. For soundness, suppose that $x \notin L$. Further assume that the first test R_1 passes. By the soundness of R_1 , it implies that (except with probability $s/2$) the string $\tilde{y}$ is $(\delta = 1/\text{polylog}(n))$ -close to some string $\tilde{y}^*$ where $(\alpha, \tilde{y}^*) \in L'$ (Note that distance in the latter statement is measured as Hamming distance between $\tilde{y}^*$ and $\tilde{y}$ as bit-strings.). By the definition of L' and of V' , it means that $\tilde{y}^*$ is the unique low-degree extensions of some table y^* , and y^* disagrees with x . For otherwise we'd have had that $V_0^x(\alpha, w) = 1$, contradicting $x \notin L$.

Let $\tilde{p}$ be the polynomial defined as in Eq. (4.1) with respect to $\tilde{y}$,²⁰ and let $\tilde{p}^*$ be defined as

$$\tilde{p}^*(\cdot) \triangleq \tilde{x}(\tilde{y}_1^*(\cdot), \dots, \tilde{y}_{m_1}^*(\cdot)) - \tilde{y}_{m_1+1}^*(\cdot).$$

Claim 4.5.1. *The bit-string representing $\tilde{p}$ is δ_1 -close to the bit-string representing $\tilde{p}^*$ in relative Hamming distance, where $\delta_1 = (m_1 + 1)(3 \log(n))\delta$.*

Proof. Since $\tilde{y}$ is δ -close to $\tilde{y}^*$, we have that

$$\Pr_{i \in [m_1+1], z \in \mathbb{F}^{m_2}} [\tilde{y}_i(z) \neq \tilde{y}_i^*(z)] \leq (3 \log(n))\delta.$$

and since both $\tilde{p}$ and $\tilde{p}^*$ compose the same $\tilde{x}$ with either $\tilde{y}_1, \dots, \tilde{y}_{m_1+1}$ or $\tilde{y}_1^*, \dots, \tilde{y}_{m_1+1}^*$ (respectively), we have

$$\Pr_{z \in \mathbb{F}^{m_2}} [\tilde{p}(z) \neq \tilde{p}^*(z)] \leq \sum_{i=1}^{m_1+1} \Pr_{z \in \mathbb{F}^{m_2}} [\tilde{y}_i(z) \neq \tilde{y}_i^*(z)] \leq (m_1 + 1) \cdot (3 \log(n))\delta.$$

Finally, the relative bit-distance between $\tilde{p}$ and $\tilde{p}^*$ is at most the relative field-symbol distance. This proves the claim. $\square$

Now, note that the total degree of $\tilde{p}^*$ is upper bounded by $d = (h-1)^2 m_1 m_2 \ll |\mathbb{F}|$ by our choices of parameters, and $\tilde{p}^*$ does not vanish on H^{m_2} since y^* disagrees with x . Fix any polynomial $q \in \text{ZoS}(\mathbb{F}, m_2, d, H)$, the polynomial $r := \tilde{p}^* - q$ is non-zero and has degree at most d . Therefore by the Schwartz-Zippel lemma,

$$\Pr_{z \in \mathbb{F}^{m_2}} [r(z) = 0] \leq \frac{d}{|\mathbb{F}|},$$

and thus the Hamming distance between $\tilde{p}^*$ and q is at least $\delta_2 = (1 - \frac{d}{|\mathbb{F}|})/(3 \log(n))$.

By triangle-inequality, the distance of $\tilde{p}$ from $\text{ZoS}(\mathbb{F}, m_2, d, H)$ is at least

$$\delta_2 - \delta_1 = \frac{1 - d/|\mathbb{F}| - (m_1 + 1)(3 \log n)^2 \delta}{3 \log n} > 1/\text{polylog}(n).$$

Therefore the second test would correctly reject with probability at least $1 - s/2$. By a union bound, the overall soundness error is at most s .

Since both R_1 and R_2 make non-adaptive queries, the overall PCP verifier also makes non-adaptive queries. To generate the accepting proof π , given witness w , one can simulate the computation $V_0^x(\alpha, w)$ and extract the relevant excerpt y by tracing the positions and values of x read in the simulation. And then compute the low-degree extensions $\tilde{y}$ from y . The relevant proofs π_1 and π_2 used for R_1 and R_2 can then be generated using $x, \tilde{y}$ and w in polynomial-time.

²⁰Since $\mathbb{F}$ is of characteristic 2, every oracle $\tilde{y}$ represents some polynomial (i.e., by representing the values of the polynomial on $\mathbb{F}^{m_2}$), albeit some $\tilde{y}$'s might represent a polynomial of high degree.

5 Plasticity for Algorithms from Hardness of Low-Space Avoid

This section is divided into four parts. In Section 5.1 we prove the main technical results, which deduce low-space AM verifiers from hardness of Avoid. In Section 5.2 we prove the main results stated in Section 1.2 as corollaries. And in Section 6.3 we prove the results stated in Section 1.3 as corollaries. Lastly in Section 5.3 we prove an equivalence result.

5.1 Simulating polytime by a low-space AM verifier

We first show that hardness of Avoid implies resource trade-offs for *sub-linear time*. Specifically, if Avoid is hard to solve in polynomial time on average (resp., hard for SearchNP on average), then any problem in $\text{DTIME}[n^\varepsilon]$ (resp., in $\text{NTIME}[n^\varepsilon]$) can be decided on average by a probabilistic verifier using polynomially less space and receiving a proof of length polynomially smaller than n^ε .

Theorem 5.1 (low-space verifier for sub-linear time from hardness of Avoid). *There is a universal constant $c > 1$ such that the following holds. Let $\mathcal{D} = \{\mathcal{D}_n\}_{n \in \mathbb{N}}$ be an ensemble of distributions. For any pair of sufficiently small constants $\varepsilon > \delta > 0$, for $m: \mathbb{N} \rightarrow \mathbb{N}$ satisfying $n^{c\varepsilon} \leq m(n) \leq n$, and let $\mathcal{C}[n^\delta \rightarrow m(n), s(n)]$ be an indexed family of circuits, where $s(\cdot)$ is any function.*

Assume that $\text{Avoid}_{\mathcal{C}}$ cannot be solved by any $\text{SearchNTIMEGuess}[\text{poly}(n), n^\varepsilon]$ algorithm with success p or more over the input distribution $\mathcal{D}$. Then, for every $L \in \text{NTIME}[n^\varepsilon]$ there is a probabilistic verifier V_L satisfying the following for every $n \in \mathbb{N}$:

1. *The verifier runs in polynomial time and space $O(n^\delta)$, and makes oracle queries to the $\mathcal{C}$ -circuit C_x indexed by its input x .²¹*
2. *The interaction is one-way, with the prover sending n^δ bits to the verifier.*
3. *With probability $1 - p$ over $x \sim \mathcal{D}_n$, if $x \in L$ then there is a prover P such that $\Pr[\langle P, V_L \rangle(x) = 1] = 1$.*
4. *Every $x \notin L$ is rejected by V_L (when interacting with any prover), with probability at least $1 - 2^{-\text{polylog}(n)}$.*

Moreover, if we only assume that $\text{Avoid}_{\mathcal{C}}$ cannot be solved by any polynomial-time algorithm with success p or more over $\mathcal{D}$ (rather than assuming hardness for SearchNTIMEGuess), then the same conclusion holds for every $L \in \text{DTIME}[n^\varepsilon]$ (rather than $\text{NTIME}[n^\varepsilon]$).

Proof. Fix a universal constant $c > 1$ such that the proof length in Theorem 4.1, when instantiated for a language in $\text{NTIME}[n^\varepsilon]$, is at most $n_0 = n^{c\varepsilon}$. We pad proofs to have length exactly n_0 . Let V_{PCP} be the verifier from Theorem 4.1, instantiated with soundness error $1/2$. Its construction uses $r(n) = \text{polylog}(n)$ random bits and makes $q_{\text{PCP}}(n) = \text{polylog}(n)$ non-adaptive queries.

A failed algorithm for Avoid. Consider the following non-deterministic algorithm A . On input $x \in \{0, 1\}^n$, it guesses a witness w for $x \in L$ and verifies it. If the verification fails, it outputs $\perp$. Otherwise, it computes the accepting PCP proof $\pi_w \in \{0, 1\}^{n_0}$ from Theorem 4.1, and outputs

$$y_w \triangleq \pi_w \circ 0^{m(n)-n_0}.$$

The algorithm A runs in polynomial time and uses at most n^ε non-deterministic bits.

Fact 5.1.1. *For every sufficiently large n , with probability at least $1 - p$ over $x \sim \mathcal{D}_n$, either $A(x)$ outputs $\perp$ on all witnesses, or there is a witness w certifying that $x \in L$ and a string $z \in \{0, 1\}^{n^\delta}$ such that $C_x(z) = y_w$.*

²¹In this model of space-bounded oracle machines, the $m(n)$ -bit answer of the oracle C_x can be scanned in read-once fashion, and storing parts of it requires using worktape space. Note that when C_x is logspace-evaluable we can simulate a space- s machine that runs in this model by a standard machine running in space $O(s + \log n)$.

Proof. By our hardness assumption, with probability at least $1 - p$ over $x \sim \mathcal{D}_n$, either $A(x)$ outputs $\perp$ on all witnesses, or $A(x)$ outputs a string $y \in \text{Range}(C_x)$ on some witness. The claim follows since the only non- $\perp$ outputs of $A(x)$ are PCP witnesses y_w corresponding to a witness w that certifies $x \in L$. $\square$

The low-space verifier for L . On input x , the prover sends $z \in \{0, 1\}^{n^\delta}$. The verifier independently samples $k = \text{polylog}(n)$ random strings $\rho_1, \dots, \rho_k \in \{0, 1\}^{r(n)}$ for V_{PCP} , and for every $j \in [k]$ it simulates V_{PCP} on input x and randomness ρ_j , using the first n_0 bits of $C_x(z)$ as its purported proof. More precisely, whenever V_{PCP} queries position $i \in [n_0]$, the verifier queries C_x on z , stores the i^{th} output bit, and uses that bit as the answer to the query. The verifier accepts if and only if all the k simulations of V_{PCP} accept.

Correctness. Consider an input $x \in L$ satisfying the condition in Fact 5.1.1. Since $x \in L$, the algorithm A outputs non- $\perp$ on some witness. Hence, there is a witness w certifying that $x \in L$ and $z \in \{0, 1\}^{n^\delta}$ such that

$$C_x(z) = \pi_w \circ 0^{m(n) - n_0}.$$

The honest prover sends this z . Every simulation of V_{PCP} then sees the accepting proof π_w , so perfect completeness of V_{PCP} implies that the verifier accepts with probability 1. This proves the claimed completeness on at least a $1 - p$ fraction of inputs.

Now fix $x \notin L$. For every fixed $z \in \{0, 1\}^{n^\delta}$, let π_z denote the first n_0 bits of $C_x(z)$. By PCP soundness, a single test using π_z accepts with probability at most $1/2$. Since the k random strings are independent, the probability that all k tests accept is at most $2^{-k} = 2^{-\text{polylog}(n)}$.

Complexity. The prover's message has n^δ bits, and the verifier stores it and simulates V_{PCP} sequentially, re-using the $\text{polylog}(n)$ bits of space. Querying C_x and storing the relevant output bit takes at most $O(\log n)$ additional space. Hence, the verifier runs in polynomial time and uses $O(n^\delta)$ space.

The “moreover” part. Suppose now that $L \in \text{DTIME}[n^\varepsilon]$. Define a deterministic polynomial-time algorithm A_0 as follows. It first decides whether $x \in L$. If $x \in L$, it computes the canonical accepting PCP proof π_x supplied by Theorem 4.1 and outputs $\pi_x \circ 0^{m(n) - n_0}$; otherwise, it outputs $0^{m(n)}$.

The hardness assumption in this case implies that, with probability at least $1 - p$ over $x \sim \mathcal{D}_n$, the output $A_0(x)$ belongs to $\text{Range}(C_x)$. On every such input belonging to L , there is a $z \in \{0, 1\}^{n^\delta}$ satisfying

$$C_x(z) = \pi_x \circ 0^{m(n) - n_0}.$$

The same verifier and the same analysis above give the claimed conclusion for every $L \in \text{DTIME}[n^\varepsilon]$. $\square$

Our next step is to “lift” Theorem 5.1 from the setting of sub-linear time to that of general polynomial time. We do this by the padding trick described in Section 2.

Remark 5.2. Observe that the proof of Theorem 5.1 gives a more fine-grained guarantee than stated. Specifically, for every $L \in \text{NTIME}[n^\varepsilon]$ there is a non-deterministic algorithm $A = A_L$ such that if A fails to solve Avoid for some low-space family $C = \{C_x\}$, then we can decide L by an AM verifier (with parameters as in the theorem’s statement). In particular, taking the contrapositive, if $L^{\text{hard}} \in \text{NTIME}[n^\varepsilon]$ cannot be solved by such an AM verifier, then A solves Avoid for every low-space family $C = \{C_x\}$.

Definition 5.3 (padded distribution). Let $\mathcal{X} = \{\mathcal{X}_n\}_{n \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n \in \mathbb{N}}$ be two ensembles of distributions, and let $T_0: \mathbb{N} \rightarrow \mathbb{N}$. We define the padded distribution $\mathcal{X}\mathcal{Y}^{T_0} = \left\{ \mathcal{X}\mathcal{Y}_N^{T_0} \right\}_{N \in \mathbb{N}}$ as follows. To draw a sample from $\mathcal{X}\mathcal{Y}_N^{T_0}$, let n be the largest integer such that $N \geq n + T_0(n)$, draw a sample $r_1 \sim \mathcal{X}_n$, draw a sample $r_2 \sim \mathcal{Y}_{N-n}$, and concatenate $r_1 \circ r_2 \in \{0, 1\}^N$.

Note that when $\mathcal{X}$ and $\mathcal{Y}$ are uniform then $\mathcal{X}\mathcal{Y}^{T_0}$ is uniform. In the following statement, we encourage the reader to first parse the text with $\mathcal{X}$ and $\mathcal{Y}$ both being uniform.

Theorem 5.4 (simulating NP by a low-space AM verifier, from hardness of Avoid). *For $\delta > 0$, let $\mathcal{X} = \{\mathcal{X}_n\}_{n \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n \in \mathbb{N}}$ be distribution ensembles such that $\mathcal{Y}$ is samplable in polynomial time and space n^δ . Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be a nice increasing function such that $T(n) \geq n$.*

Let $c > 1$ be the universal constant from Theorem 5.1. Assume that for a sufficiently small $\varepsilon > \delta$, there is a function $m : \mathbb{N} \rightarrow \mathbb{N}$ satisfying $n^{c\varepsilon} \leq m(n) \leq n$ and an indexed family $\mathcal{C}[n^\delta \rightarrow m(n), s(n)]$ such that $\text{Avoid}_{\mathcal{C}}$ cannot be solved by any $\text{SearchNTimeGuess}[\text{poly}(n), n^\varepsilon]$ algorithm with success $p(n)$ or more over the input distribution $\mathcal{X}\mathcal{Y}^{T^{1/\varepsilon}}$, where s is any function.

Then, for every $L \in \text{NTIME}[T]$ there is an AM verifier V_L that successfully decides L with probability $(1 - p)$ over the input distribution $\mathcal{X}$ and its internal randomness, with the following properties:

1. *The verifier runs in time $T^{O(1/\varepsilon)}$ and space $O(T^{\delta/\varepsilon})$ (not counting the space for storing randomness), and makes oracle queries to the $\mathcal{C}$ -circuit C_{xy} where x is its input and y is a random sample from $\mathcal{Y}_{T(n)^{1/\varepsilon}}$.²² In addition, the verifier stores its randomness on a special read-only tape to which it has two-way access, and the contents of this special tape are not counted towards the verifier's space complexity.*
2. *There is 1 round (2 messages) of interaction, and the size of the honest prover's messages is $T^{\delta/\varepsilon}$ bits.*
3. *With probability $1 - p(N)$ over $x \sim \mathcal{X}_n$ and over the internal randomness of V_L , where $N = n + T(n)^{1/\varepsilon}$, if $x \in L$ then there are prover messages causing V_L to accept x .*
4. *Every $x \notin L$ is rejected by V_L (when interacting with any prover), with probability at least $1 - 2^{-\text{polylog}(N)}$.*

Moreover, if we only assume that $\text{Avoid}_{\mathcal{C}}$ cannot be solved by any polynomial-time algorithm with success p or more over $\mathcal{X}\mathcal{Y}^{T^{1/\varepsilon}}$ (rather than assuming hardness for SearchNTimeGuess), then the same conclusion holds for every $L \in \text{DTIME}[T]$ (rather than $\text{NTIME}[T]$).

Proof. We prove the basic part of the statement, and omit the proof of the “moreover” part, which is near-identical. We instantiate Theorem 5.1 with $\mathcal{D} = \mathcal{X}\mathcal{Y}^{T^{1/\varepsilon}}$ and ε, δ and $\mathcal{C}$, to deduce that for every $L \in \text{NTIME}[n^\varepsilon]$ there is a verifier V_0 as stated there.

We first transform V_0 into a verifier V that sends a challenge to the prover and then receives a response (at which point it decides deterministically). To do so we use the standard trick of Babai and Moran [BM88], exploiting the fact that in our case the prover's message is short. Specifically, recall that V_0 receives a message of length n^δ and uses $\text{polylog}(n)$ random coins. The verifier V independently draws $k = n^\delta + \text{polylog}(n)$ random choices $r_1, \dots, r_k$ for V_0 , expects a response $z \in \{0, 1\}^{n^\delta}$, and accepts if and only if V_0 accepts z with all of $r_1, \dots, r_k$. When $x \notin L$, by a union-bound over all $z \in \{0, 1\}^{n^\delta}$, the probability that there is z such that V accepts is at most $2^{n^\delta} \cdot 2^{-k} = 2^{-\text{polylog}(n)}$. The verifier V still runs in polynomial time and $O(n^\delta)$ space, but now also uses $k \cdot \text{polylog}(n)$ bits of persistent randomness (for storing $r_1, \dots, r_k$).

Fixing $L \in \text{NTIME}[T]$, we define a padded language L' as follows. On input length N , let n be the largest integer such that $N \geq n + T(n)^{1/\varepsilon}$. Parse the input as $xy \in \{0, 1\}^N$ where $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^{N-n}$. Then, $xy \in L'$ if and only if $x \in L$. Note that $L' \in \text{NTIME}[T(|x|)] \subseteq \text{NTIME}[N^\varepsilon]$. Hence, there is an AM verifier $V_{L'}$ for L' as above.

We obtain an AM verifier V_L for L as follows: given x , the verifier V_L uses its randomness to sample $y \sim \mathcal{Y}_{T(n)^{1/\varepsilon}}$, includes y as part of its challenge to the prover, and simulates $V_{L'}$ on virtual input xy . Whenever $V_{L'}$ accesses its input, V_L uses its two-way access to its internal randomness to compute y again and answer.

Denoting $N = |xy| = n + T(n)^{1/\varepsilon} = O(T^{1/\varepsilon})$, and plugging input length N into the time and space and communication bounds on $V_{L'}$, we deduce the corresponding bounds for V_L . The time and space complexity need to also account for the cost of re-computing y , which adds a $\text{poly}(T^{1/\varepsilon}) = \text{poly}(T)$ multiplicative overhead to the time and an $O(T^{\delta/\varepsilon})$ additive overhead to the space. Also, $V_{L'}$ makes queries to the circuit indexed by its input xy , which is C_{xy} for a random $y \sim \mathcal{Y}_{T^{1/\varepsilon}}$, as stated.

²²In this model of space-bounded oracle machines, the $m(|xy|)$ -bit answer of the oracle C_{xy} can be scanned in read-once fashion, and storing parts of it requires using worktape space. Note that when C_{xy} is logspace-evaluable we can simulate a space- s machine that runs in this model by a standard machine running in space $O(s + \log T)$.

Finally, we know that $V_{L'}$ rejects every $xy \notin L'$ with probability at least $1 - 2^{-\text{polylog}(N)}$, hence V_L rejects every $x \notin L$ with such probability. And with probability at least $1 - p(N)$ over $xy \sim \mathcal{XY}^{T^{1/\varepsilon}}$, if $x \in L$ then there is a prover causing $V_{L'}$ to accept with probability 1. Hence, with probability $1 - p(N)$ over $x \sim \mathcal{X}$ and over the random coins of V_L , which yield a sample $xy \sim \mathcal{XY}^{T^{1/\varepsilon}}$, if $x \in L$ then there are prover messages causing V_L to accept. $\square$

Remark 5.5. Similarly to Remark 5.2, the proof above gives a more fine-grained guarantee: For every $L \in \text{NTIME}[T]$ there is a non-deterministic $A = A_L$ such that if A fails to solve Avoid for some low-space $C = \{C_x\}$, then we can decide L by a corresponding AM verifier.

In Theorems 5.1 and 5.4 the AM verifier made queries to the Avoid family C_x . If the hard Avoid family is evaluable in simultaneous low space and polynomial time, then the AM verifier can evaluate C_x by itself. This yields Theorem 2.1, which we now prove.

Theorem 5.6 (Theorem 2.1, restated). *Suppose that Assumption 1.1 holds for $C = \{C_x\}$ that is evaluable in simultaneous time $\text{poly}(n)$ and space n^{ε^2} . Then for every $L \in \text{P}$ and every constant $\gamma > 0$ there is an AM verifier V with the following properties:*

- **Complexity:** *The verifier V runs in time $\text{poly}(n)$, uses at most n^γ space, and has prover message length n^γ . It also uses $\text{poly}(n)$ bits of “persistent” randomness (i.e., the randomness is stored on a special tape to which V has two-way read-only access).*
- **Average-case completeness.** *With probability at least $1 - p(n^{O(1)}) - 2^{-\text{polylog}(n)}$ over a choice of $x \in \{0, 1\}^n$ and over V ’s random coins, if $x \in L$ then there is a prover message causing V to accept.*
- **Worst-case soundness.** *For every $x \notin L$, the probability over V ’s random coins that there is a prover message causing V to accept is at most $2^{-\text{polylog}(n)}$.*

Proof. Let $\gamma > 0$ and let $L \in \text{DTIME}[n^k]$. Let $c > 1$ be the universal constant from Theorem 5.4, and choose a sufficiently small constant $\theta > 0$ such that $ck\theta < \gamma$. By Assumption 1.1, there is a family $C = \{C_x\}$ evaluable in simultaneous polynomial time and space n^{θ^2} , stretching n^{θ^2} bits to n^θ bits, that cannot be solved in polynomial time with success p or more over the uniform distribution of inputs x .

We apply Theorem 5.4 with parameters $\varepsilon = \theta/c$ and $\delta = \theta^2$. This gives an AM verifier V_0 . The verifier V simulates V_0 and resolves its queries to C_{xy} in polynomial time. Both this evaluation and the simulation of V_0 use space

$$O\left((n^k)^{\delta/\varepsilon}\right) = O(n^{ck\theta}) = O(n^\gamma),$$

using the persistent access of V to y , which is part of its randomness, and reusing space across queries. The prover message length is bounded by the same quantity. $\square$

5.2 Consequences I: Trading time for space and alternations for time

We now prove implications of the technical results from Section 5.1, and in particular we prove the main results that were stated in the introduction. Throughout the exposition in this section, when we say “Avoid is hard”, we mean hardness for an indexed family that is evaluable in low space.

We start with an implication that follows using the assumption in Theorem 5.4 alone (i.e., without additional assumptions). We show that if Avoid is hard on average then every problem in P can be simulated in small probabilistic space where the low-space machine has persistent two-way access to its randomness.

Corollary 5.7 (trading time for space with persistent randomness). *Suppose that the hardness assumption in the “moreover” part of Theorem 5.4 holds with respect to $\mathcal{X}, \mathcal{Y}, \varepsilon, \delta$ and $\mathcal{C}$, and in addition assume that $\mathcal{C}$ is evaluable in space $s(n)$. Then for every $L \in \text{DTIME}[T]$ there is a $\text{BP}^*\text{SPACE}[O(T^{\delta/\varepsilon} + s(T^{1/\varepsilon}))]$ machine*

M_L such that

$$\Pr_{x \sim \mathcal{X}_n} \left[\Pr_{M_L} [M_L(x) = L(x)] \geq 2/3 \right] \geq 1 - \mu ,$$

where $\mu = 3(p(N) + 2^{-\text{polylog}(N)})$ and $N = O(n + T^{1/\varepsilon})$.

Proof. Let $L \in \text{DTIME}[T]$, and let V_L be the verifier guaranteed by Theorem 5.4. Over a random choice of $x \sim \mathcal{X}_n$ and random coins for V_L , denote by $\mathcal{E}$ the event that

$x \in L$ and there's no prover message convincing $V_L \bigvee x \notin L$ and there is a prover message convincing V_L .

Since $\Pr_{x \sim \mathcal{X}_n, V_L} [\mathcal{E}] \leq p(N) + 2^{-\text{polylog}(N)} = \mu/3$, Markov's inequality gives

$$\Pr_{x \sim \mathcal{X}_n} \left[\Pr_{V_L} [\mathcal{E}] > 1/3 \right] \leq \mu .$$

Thus, for all but a μ fraction of the inputs, the probability (over V_L 's coins) of $\mathcal{E}$ is at most $1/3$.

Denote the total prover communication by s_1 and the work usage of V_L (excluding its read-only two-way random tape) by s_2 . We define a probabilistic machine M that on input x and fixed randomness r enumerates all 2^{s_1} possible prover messages and accepts iff there is a message sequence causing V_L to accept x when using randomness r . The space usage of M (excluding the persistent randomness) is $O(s_1 + s_2)$, and by the above it correctly decides L (i.e., with error probability at most $1/3$) on at least $1 - \mu$ of inputs. $\square$

We next prove Theorem 1.2, which relies on hardness of Avoid as well as on the additional assumption that $\text{prBP}^*\text{L} = \text{prL}$.

Theorem 5.8 (Theorem 1.2, restated). *Suppose that Assumption 1.1 holds and that $\text{prBP}^*\text{L} = \text{prL}$. Then for every $L \in \text{P}$ and every $\varepsilon > 0$ there is a space- n^ε algorithm A such that $\Pr_{x \in \{0,1\}^n} [A(x) = L(x)] \geq 1 - 3(p(n^{O(1)}) + 2^{-\text{polylog}(n)})$ for all $n \in \mathbb{N}$.*

Proof. Let $\gamma > 0$ and let $L \in \text{DTIME}[T]$, where $T(n) = n^k$ for some constant k . Let $c > 1$ be the universal constant from Theorem 5.4, and choose a sufficiently small constant $\theta > 0$ such that $ck\theta < \gamma$. Set $\varepsilon = \theta/c$ and $\delta = \theta^2$. By Assumption 1.1, there is a family $C = \{C_x\}$ evaluable in space n^{θ^2} , stretching n^{θ^2} bits to n^θ bits, that cannot be solved in polynomial time with success p over the uniform distribution of inputs.

By Corollary 5.7, there is a $\text{BP}^*\text{SPACE}[S]$ machine M_L deciding L on all but at most

$$\mu = 3\left(p(O(n + T^{1/\varepsilon})) + 2^{-\text{polylog}(O(n + T^{1/\varepsilon}))}\right) = 3\left(p(\text{poly}(n)) + 2^{-\text{polylog}(n)}\right)$$

fraction of the inputs, where

$$S = O\left(T^{\delta/\varepsilon} + (T^{1/\varepsilon})^{\theta^2}\right) = O(n^{ck\theta}) = O(n^\gamma) .$$

Since $\text{prBP}^*\text{L} = \text{prL}$, by a standard padding argument it follows that $\text{prBP}^*\text{SPACE}[O(n^\gamma)] = \text{prDSPACE}[O(n^\gamma)]$. Applying the latter statement to the promise-problem decided by M_L (i.e., the promise includes the $1 - \mu$ fraction of inputs on which M_L correctly decides L), there is a deterministic space- $O(n^\gamma)$ machine deciding L correctly with probability at least $1 - \mu$ over a uniformly chosen input. $\square$

We now prove Theorem 1.3. Recall that the derandomization assumption in this result is stronger: instead of $\text{prBP}^*\text{L} = \text{prL}$, we now assume lower bounds against circuits with SAT oracles (as in [KvM02]). The conclusion will be linear-time simulation of P in the polynomial-time hierarchy.

Theorem 5.9 (Theorem 1.3, restated). *Suppose that Assumption 1.1 holds with $C = \{C_x\}$ that is evaluable in simultaneous time $\text{poly}(n)$ and space n^{ε^2} . Also assume that $\text{DSPACE}[O(n)] \notin \text{i.o.SIZE}^{\text{SAT}}[2^{\delta n}]$ for some $\delta > 0$. Then for every $L \in \text{P}$ there is a constant $k = k_L$ and a linear-time Σ_k machine that decides L correctly on at least $1 - 3(p(n^{O(1)}) + 2^{-\text{polylog}(n)})$ of the n -bit inputs.*

Proof. Let $L \in \text{DTIME}[n^k]$, and let $c > 1$ be the universal constant from Theorem 5.4. Choose a sufficiently small constant $\theta > 0$ such that $ck\theta < 1/2$, and set $\varepsilon = \theta/c$ and $\delta = \theta^2$. Relying on Assumption 1.1, let $C = \{C_x\}$ be the family for the parameter θ . We apply Theorem 5.4 with parameters ε and δ . We deduce that there is an AM verifier V deciding L correctly with probability $1 - \mu$, where $\mu = p(n^{O(1)}) + 2^{-\text{polylog}(n)}$ (over a choice of input and its internal coins), such that V uses polynomial time n^C , space at most $n^{ck\theta} < \sqrt{n}$, and read-only two-way access to its random coins. (Note that the time bound of n^C used the fact that C_{xy} is evaluable in time $\text{poly}(|xy|)$.) As in the proof of Corollary 5.7, we define $\mathcal{E}$ to be the event that

$$x \in L \text{ and there's no prover message convincing } V \vee x \notin L \text{ and there is a prover message convincing } V.$$

and observe by Markov's inequality that, for all but a 3μ fraction of the inputs, the probability of $\mathcal{E}$ over V 's random coins is at most $1/3$.

Relying on the derandomization assumption, there is a PRG stretching $\ell = O(\log n)$ bits to n^C bits that is computable in space $O(\log n)$ and $(1/n)$ -fools SAT-oracle circuits of size n^{2C} (see [KvM02]). We define a deterministic NP-verifier V' that receives 2^ℓ messages (in parallel) from the prover, simulates V with all 2^ℓ choices of random coins generated by the PRG, and accepts if and only if for at least .6 of these choices, the corresponding message by the prover convinces V to accept. Note that in each of the 2^ℓ invocations of V , each query to its random coins can be answered by re-computing the PRG on-the-fly (on the corresponding ℓ -bit seed); hence, the time complexity of V' is $\text{poly}(n)$ and its space complexity is $O(\sqrt{n} + \log n) = O(\sqrt{n})$. Also, on each input x , the probability of $\mathcal{E}$ under the PRG is $(1/n)$ -close to its probability under a uniform string,²³ and hence V' decides L correctly on all but at most a 3μ fraction of the inputs.

Let $L' \in \text{NTISP}[\text{poly}(n), O(\sqrt{n})]$ be the language decided by V' . By [Nep70], we have $L' \in \Sigma_k \text{TIME}[O(n)]$ for a sufficiently large constant k . $\square$

Finally, we prove Theorem 1.4, whose conclusion is worst-case simulation of exponential-time algorithms. Specifically, Theorem 1.4 follows from the two statements below along with Theorems 5.8 and 5.9.

Theorem 5.10 (worst-case simulation of E in low space). *Assume that for every $L \in \text{P}$ and every $\varepsilon > 0$ there is a space- n^ε algorithm A such that $\Pr_{x \in \{0,1\}^n}[A(x) = L(x)] \geq .999$ for all $n \in \mathbb{N}$. Then*

$$\text{E} \subseteq \bigcap_{\varepsilon > 0} \text{SPACE}[2^{\varepsilon n}].$$

Proof. For the proof we will need a systematic error-correcting code ECC mapping N bits to $\text{poly}(N)$ bits in time $\text{poly}(N)$ that is locally decodable from .01 errors. Suitable codes are standard, for example concatenating the Reed-Muller code with the Hadamard code (see, e.g., [Vad12, Theorem 7.51]).

Fixing any $L \in \text{E}$, let $\bar{L}$ be the language whose truth-tables are encodings of the truth-tables of L .²⁴ For a small enough $\delta = \delta(\varepsilon) > 0$, let $\bar{L}^{\text{pad}} = \{xp : x \in \bar{L} \wedge |p| = 2^{\delta \cdot |x|}\}$, and note that $\bar{L}^{\text{pad}} \in \text{P}$. Hence, there is a space- $O(N)$ machine A deciding $\bar{L}^{\text{pad}}$ correctly on at least 0.999 of the N -bit inputs, for every $N \in \mathbb{N}$.

Given $x \in \{0,1\}^n$, to decide if $x \in L$ we run the local decoder for $\bar{L}$ on the corresponding input $\bar{x} \in \{0,1\}^{\bar{n}=O(n)}$ (i.e., on the $\bar{n}$ -bit input $\bar{x}$ such that $f_x = \text{ECC}(f)_{\bar{x}}$) and whenever the decoder queries the corrupt codeword at location $q \in \{0,1\}^{\bar{n}}$, we draw a random $p \in \{0,1\}^{2^{\delta \cdot |q|}}$, and run the machine A on (virtual) input $\bar{q} = qp$. To facilitate the analysis, we consider an algorithm that acts as above, but draws a single random p to use across all queries q (rather than drawing an independent p for every query q).

Since A is correct on 0.999 of the inputs of length $\bar{n} + 2^{\delta \bar{n}}$, we have

$$0.001 \geq \Pr_{qp}[A(qp) \neq \bar{L}^{\text{pad}}(qp)] = \mathbb{E}_p[\Pr_q[A(qp) \neq \bar{L}^{\text{pad}}(qp)]] = \mathbb{E}_p[\Pr_q[A(qp) \neq \bar{L}(q)]] ,$$

and thus with probability at least 9/10 over p we have that $\Pr_q[A(qp) \neq \bar{L}(q)] \leq .01$. Whenever this happens, the decoder in the algorithm above is given access to a corrupt codeword that agrees with $\bar{L}$ on 0.99 of the $\bar{n}$ -bit inputs, in which case the decoder succeeds with high probability.

²³This is because we can decide $\mathcal{E}$ (on an input random string) using a circuit of size $\text{poly}(n)$ with SAT-gates.

²⁴That is, for every $n \in \mathbb{N}$, let $L_n \in \{0,1\}^{2^n}$ be the truth-table of L on n -bit inputs, we define $\bar{L}_{\bar{n}=O(n)} = \text{ECC}(L_n) \in \{0,1\}^{2^{\bar{n}=O(n)}}$ to be the truth-table of $\bar{L}$ on $\bar{n}$ -bit inputs. On all other input lengths $\bar{L}$ is defined trivially.

The space complexity of the algorithm is dominated by the space complexity of A and the space complexity of storing $p \in \{0, 1\}^{2^{\delta \cdot \bar{n}}}$. This is since the local decoder runs in time (and space) $\text{polylog}(N)$, and uses space-efficient (“emulative”) composition of space-bounded algorithms. We run A on inputs of length $\bar{n} + 2^{\delta \cdot \bar{n}}$, and since δ is small enough A runs in space $2^{(\varepsilon/2) \cdot n}$. Similarly, storing p requires space $2^{\delta \cdot \bar{n}} < 2^{(\varepsilon/2) \cdot n}$.

The above implies that $L \in \text{BPSPACE}[2^{(\varepsilon/2) \cdot n}]$. Using the classical unconditional derandomization by Nisan [Nis92] (with appropriate padding), it follows that $L \in \text{SPACE}[2^{\varepsilon \cdot n}]$. $\square$

Theorem 5.11 (worst-case simulation of $\mathbf{E}$ in small time with alternations). *If for every $L \in \mathbf{P}$ there is $k = k_L$ and a linear-time Σ_k machine that decides L correctly on at least .999 of the n -bit inputs, then*

$$\mathbf{E} \subseteq \bigcap_{\varepsilon > 0} \bigcup_{k \in \mathbb{N}} \Sigma_k \text{TIME}[2^{\varepsilon \cdot n}].$$

Proof. We use the same code ECC from the proof of Theorem 5.10, and define $\bar{L}$ and $\bar{L}^{\text{pad}}$ identically. Let A be a Σ_k -machine running in time $O(N)$ that decides $\bar{L}^{\text{pad}}$ on 0.999 of the N -bit inputs. The algorithm for L is identical to the one from the proof of Theorem 5.10, and its complexity is dominated by the complexity of running A on inputs of length $O(2^{\delta \cdot n})$, hence the algorithm runs in time less than $2^{\varepsilon \cdot n}$.

The only issue is that so far we obtained a randomized algorithm, which first tosses $2^{\delta \cdot n}$ coins and then uses k alternations. To eliminate the random choice of coins we use more alternations, imitating the textbook simulation of BPP in Σ_2 but with a Σ_k -predicate that follows the coin tosses.²⁵ $\square$

5.3 Consequences II: An equivalence

In this subsection, we show that, assuming $\text{prBP}^* \mathbf{L} = \text{prL}$, average-case low-space simulation of polynomial-time functions is equivalent to average-case hardness of Avoid. We also give an equivalent formulation in terms of a natural low-space diagonalization problem.

The key technical ingredient is a proof-printing refinement of Theorem 5.1. For each suitable sublinear-time language L , it fixes an Avoid algorithm A independently of the circuit family $\mathcal{C}$, and gives a low-space machine M such that for every x , either $M(x)$ prints a full PCP proof that $x \in L$ or $A(x)$ solves $\text{Avoid}_{\mathcal{C}}$ on x . Thus, average-case hardness of Avoid implies that M prints a PCP proof for most x 's. By choosing the PCP proof to contain the output of a fixed computation, we turn this reconstruction into multi-bit function simulation.

Theorem 5.12 (low-space PCP proof for sub-linear time from hardness of Avoid). *There is a universal constant $c > 1$ such that the following holds. For every pair of sufficiently small constants $\varepsilon > \delta > 0$, every $L \in \text{NTIME}[n^\varepsilon]$, and every $q: \mathbb{N} \rightarrow \mathbb{N}$ satisfying $n^{c\varepsilon} \leq q(n) \leq n$, there is a non-deterministic polynomial-time algorithm $A = A_{L, \varepsilon, \delta, q}$ that uses at most n^ε non-deterministic bits, outputs a $q(n)$ -bit string on each accepting path, and satisfies the following.*

For every PCP verifier V whose parameters match those in Theorem 4.1, and every indexed family $\mathcal{C}[n^\delta \rightarrow q(n), s(n)]$ evaluable in space $O(n^\delta)$, where $s(\cdot)$ is any function, there is a deterministic machine $M_{L, \mathcal{C}}$ using space $O(n^\delta)$ such that, for every sufficiently large n and every $x \in \{0, 1\}^n$:

1. *If $M_{L, \mathcal{C}}(x) \neq \perp$, then $M_{L, \mathcal{C}}(x)$ outputs a PCP proof π_x such that $V^{\pi_x}(x)$ accepts with probability 1.*
2. *If $x \notin L$, then $M_{L, \mathcal{C}}(x) = \perp$.*
3. *If $x \in L$ and $M_{L, \mathcal{C}}(x) = \perp$, then $A(x)$ solves $\text{Avoid}_{\mathcal{C}}$.*

²⁵In more detail, think of the initial procedure as tossing $T = 2^{\delta \cdot n}$ coins and then applying a deterministic (Σ_k) predicate P . Using randomness-efficient samplers as in [GZ11], at the cost of a polynomial blow-up in the number of random coins (and running time) from T to $\bar{T} = \text{poly}(T)$, we may assume that this algorithm errs on at most $2^{\bar{T} \cdot 0.1}$ choices of random coins. For $\bar{k} = k + 2$, the $\Sigma_{\bar{k}}$ machine for L decides whether there is a prefix of $\bar{T}/2$ coins such that for all suffixes of $\bar{T}/2$ coins the original Σ_k machine accepts (with these coins). The correctness follows the standard analysis (see [GZ11]), and the running time is still less than $2^{\varepsilon \cdot n}$ if we choose $\delta > 0$ to be sufficiently small (to compensate for the additional polynomial overhead caused by the sampler-based error reduction).

Consequently, let $\mathcal{D} = \{\mathcal{D}_n\}_{n \in \mathbb{N}}$ be any ensemble of distributions such that $\mathcal{D}_n$ is supported on $\{0, 1\}^n$, and let $p: \mathbb{N} \rightarrow [0, 1]$. If, for every sufficiently large n ,

$$\Pr_{x \sim \mathcal{D}_n} [A(x) \text{ solves Avoid}_{\mathcal{C}}] \leq p(n) ,$$

then

$$\Pr_{x \sim \mathcal{D}_n} [x \in L \text{ and } M_{L, \mathcal{C}}(x) = \perp] \leq p(n) .$$

Moreover, if $L \in \text{DTIME}[n^\varepsilon]$, then A can be made deterministic and total, outputting a $q(n)$ -bit string on every n -bit input.

Proof. Fix a PCP system (e.g. the one in Theorem 4.1) and a universal constant $c > 1$ such that the PCP proof length, when instantiated for a language in $\text{NTIME}[n^\varepsilon]$, is at most $n_0 = n^{c\varepsilon}$. We pad proofs to have length exactly n_0 . Let V_{PCP} be the PCP verifier with soundness error $1/2$, using $r(n) = \text{polylog}(n)$ random bits and $\text{polylog}(n)$ space.

We first define $A = A_{L, \varepsilon, \delta, q}$. On input $x \in \{0, 1\}^n$, the algorithm guesses a witness w for $x \in L$ and verifies it. If the verification fails, it rejects. Otherwise, it computes the corresponding accepting PCP proof $\pi_w \in \{0, 1\}^{n_0}$ and outputs

$$y_w \triangleq \pi_w \circ 0^{q(n) - n_0} .$$

Thus, A runs in polynomial time, uses at most n^ε non-deterministic bits, and outputs a $q(n)$ -bit string on every accepting path.

Fix an indexed family $\mathcal{C}$ as in the statement, and set $\ell \triangleq n^\delta$. The deterministic machine $M_{L, \mathcal{C}}$ enumerates all $z \in \{0, 1\}^\ell$. For each z , let $\pi_z \in \{0, 1\}^{n_0}$ be the first n_0 bits of $C_x(z)$. The machine enumerates all random strings $\rho \in \{0, 1\}^{r(n)}$ and simulates $V_{\text{PCP}}^{\pi_z}(x, \rho)$. Whenever the PCP verifier queries position $i \in [n_0]$, the machine computes the i th output bit of $C_x(z)$ using the space-bounded evaluation algorithm for $\mathcal{C}$.

If some simulation of V_{PCP} rejects, the machine proceeds to the next z . If every simulation accepts, it prints π_z one bit at a time, reevaluating the corresponding output bits of $C_x(z)$, and halts. If no z passes the test, then it outputs $\perp$.

Correctness. Whenever $M_{L, \mathcal{C}}(x)$ prints a string π_z , the PCP verifier accepts it for every random string, and hence π_z is a PCP proof witnessing that $x \in L$. Also, it follows that $M_{L, \mathcal{C}}(x) = \perp$ for every $x \notin L$.

Now suppose that $x \in L$ and $M_{L, \mathcal{C}}(x) = \perp$. The algorithm A has at least one accepting path. Moreover, for every accepting witness w , we have $y_w \notin \text{Range}(C_x)$. For otherwise some $z \in \{0, 1\}^\ell$ would satisfy

$$C_x(z) = \pi_w \circ 0^{q(n) - n_0} .$$

The first n_0 bits of $C_x(z)$ would then be the honest proof π_w , which is accepted for every random string, and would have been printed by $M_{L, \mathcal{C}}$, contradicting $M_{L, \mathcal{C}}(x) = \perp$. Hence every accepting path of $A(x)$ outputs an $\text{Avoid}_{\mathcal{C}}$ solution for x .

In particular, if $A(x)$ solves $\text{Avoid}_{\mathcal{C}}$ on at most $p(n)$ fraction of inputs x , then for at most $p(n)$ inputs x it holds that $x \in L$ and $M_{L, \mathcal{C}}(x) = \perp$.

Complexity. For space usage, the machine $M_{L, \mathcal{C}}$ stores z , a PCP random string, and the indices of its enumerations. It simulates the PCP verifier in $\text{polylog}(n)$ space and evaluates requested output bits of $C_x(z)$ in space $O(n^\delta)$, reusing this workspace across all choices of z and ρ and while printing the proof. Since $\ell = n^\delta$, its total space usage is

$$O(n^\delta + \ell + \text{polylog}(n)) = O(n^\delta) .$$

The “moreover” part. Suppose now that $L \in \text{DTIME}[n^\varepsilon]$. Define A deterministically where it first decides whether $x \in L$. If $x \in L$, it computes the canonical accepting PCP proof π_x and outputs $\pi_x \circ 0^{q(n)-n_0}$; otherwise, it outputs $0^{q(n)}$. Such algorithm A is deterministic and total. The correctness and space complexity follow from the same argument as above. $\square$

We are now ready to prove the equivalence in Theorem 1.2. As mentioned above, we will also prove equivalence of the two statements to a third statement, which asserts that a certain natural low-space function is hard for FP. We define this function and then state and prove the result.

Definition 5.13 (low-space diagonalization). *For every constant $\varepsilon \in (0, 1)$, let $m_\varepsilon(n) = \lceil n^{\varepsilon^2} \rceil$ and fix a universal deterministic machine U_ε . On input $(x, d) \in \{0, 1\}^n \times \{0, 1\}^{m_\varepsilon(n)}$, U_ε parses d as the description of a deterministic machine, simulates that machine on x for at most $2^{n^{\varepsilon^2}}$ steps and using at most n^{ε^2} work space, and returns its n -bit output (or 0^n if the simulation exceeds the time or space bounds, or returns an output of other length). Define the diagonalization problem as*

$$\text{Diag}_\varepsilon(x) = \left\{ y \in \{0, 1\}^n : \forall d \in \{0, 1\}^{m_\varepsilon(n)}, y \neq U_\varepsilon(x, d) \right\}.$$

In particular, $\text{Diag}_\varepsilon(x)$ asks for a string that differs from the output on x of every low-space machine. Note that the problem is total for all sufficiently large n , and U_ε can be evaluated in space $O(n^{\varepsilon^2})$.

Theorem 5.14 (Theorem 1.2, a more general version). *Assume that $\text{prBP}^*\text{L} = \text{prL}$. Then the following are equivalent:*

1. *For every $f \in \text{FP}$ and every constant $\gamma > 0$, there is a deterministic space- n^γ machine M_f such that, for every sufficiently large n ,*

$$\Pr_{x \in \{0, 1\}^n} [M_f(x) = f(x)] \geq 1 - n^{-\omega(1)}.$$

2. *Assumption 1.1 holds with $p(n) = n^{-\omega(1)}$.*

3. *For every constant $\varepsilon \in (0, 1)$ and every deterministic polynomial-time machine B , for every sufficiently large n ,*

$$\Pr_{x \in \{0, 1\}^n} [B(x) \in \text{Diag}_\varepsilon(x)] \leq n^{-\omega(1)}.$$

Proof. We prove that Item (1) implies Item (3), Item (3) implies Item (2), and Item (2) implies Item (1).

(1) $\Rightarrow$ (3). Suppose that Item (1) holds. Fix ε and a machine B as in Item (3), and apply Item (1) to $B(x)$ with $\gamma = \varepsilon^2$. Let M_B be the resulting space- n^{ε^2} machine and let d_B be its description. For every sufficiently large n we have $U_\varepsilon(x, d_B) = M_B(x)$, and hence

$$\Pr_{x \in \{0, 1\}^n} [B(x) \in \text{Diag}_\varepsilon(x)] \leq \Pr_{x \in \{0, 1\}^n} [B(x) \neq M_B(x)] \leq n^{-\omega(1)}.$$

(3) $\Rightarrow$ (2). We next prove that Item (3) implies Item (2). Fix ε , and for every $x \in \{0, 1\}^n$ define $C_x: \{0, 1\}^{m_\varepsilon(n)} \rightarrow \{0, 1\}^{n^\varepsilon}$ such that $C_x(d)$ outputs the first n^ε bits of $U_\varepsilon(x, d)$. Let $\mathcal{C} = \{C_x\}_{x \in \{0, 1\}^n}$, and observe that this family is evaluable in space $O(n^{\varepsilon^2})$.

For every deterministic polynomial-time machine A , define $B_A(x) = A(x) \circ 0^{n-n^\varepsilon}$. If $A(x)$ solves $\text{Avoid}_{\mathcal{C}}$, then $B_A(x) \in \text{Diag}_\varepsilon(x)$. Therefore, by Item (3),

$$\Pr_{x \in \{0, 1\}^n} [A(x) \text{ solves } \text{Avoid}_{\mathcal{C}}] \leq \Pr_{x \in \{0, 1\}^n} [B_A(x) \in \text{Diag}_\varepsilon(x)] \leq n^{-\omega(1)}.$$

Thus, $\mathcal{C}$ serves as the hard family of Avoid instances for Assumption 1.1.

(2) $\Rightarrow$ (1). We finally prove that Item (2) implies Item (1). Fix $f \in \text{FP}$ computable by an algorithm F in time n^k , let L be the language that simulates F and accepts and let $\gamma > 0$.

Let $c > 1$ be a sufficiently large universal constant for the construction in Theorem 5.12 and for the PCP used below. Choose a sufficiently small constant $\theta > 0$ such that $ck\theta < \gamma$. Set $\alpha = \theta/c$ and $\delta = \theta^2$.

We now define a padded language $L^{\text{pad}} = \{(x, y) : x \in L \wedge |x| \leq |(x, y)|^{\alpha/k}\}$, and note that $L^{\text{pad}} \in \text{DTIME}[N^\alpha]$. We will apply Theorem 5.12 to L^{pad} , using a slightly augmented PCP that reveals the computational history of L^{pad} , which in particular contains $f(x)$.

Specifically, let V_0 be the verifier from Theorem 4.1 for L^{pad} . The new proof is a pair (π, τ) , where π is a proof for V_0 and τ is a standard locally checkable tableau of the computation of $F(x)$. The verifier runs $V_0^\pi(x, y)$ and, using additional $k \log n$ bits of randomness, checks one local constraint of τ . It accepts if and only if both tests accept. This verifier retains perfect completeness and the soundness of V_0 , remains non-adaptive, and makes $\text{polylog}(N)$ queries using $\text{polylog}(N)$ space. Its proof length is at most $N^{c\alpha} = N^\theta$, and an honest PCP proof is computable in polynomial time from x . If a proof is accepted for every verifier random string, then τ contains $f(x)$.

Let A be the deterministic polynomial-time machine obtained from Theorem 5.12 using the PCP above. By Item (2), for the parameter θ there is an indexed family $\mathcal{C}[N^{\theta^2} \rightarrow N^\theta, s(N)]$ evaluable in space $O(N^{\theta^2})$, satisfying that

$$p(N) \triangleq \Pr_{z \in \{0,1\}^N} [A(z) \text{ solves Avoid}_{\mathcal{C}}] \leq N^{-\omega(1)}.$$

As such, the corresponding proof-printing machine R from Theorem 5.12 uses space $O(N^\delta)$ and satisfies

$$\Pr_{z \in \{0,1\}^N} [z \in L^{\text{pad}} \text{ and } R(z) = \perp] \leq p(N).$$

Let $\ell = \lceil n^{k/\alpha} \rceil$ and $N = n + \ell$. For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^\ell$, the string xy belongs to L^{pad} , and every non- $\perp$ output of $R(xy)$ contains the desired output $f(x)$. We have,

$$\Pr_{\substack{x \in \{0,1\}^n \\ y \in \{0,1\}^\ell}} [R(xy) = \perp] \leq p(N) \leq n^{-\omega(1)}.$$

Thus, using y as persistent randomness and simulating $R(xy)$ gives us a BP^*SPACE machine that outputs either $f(x)$ or $\perp$ on every input (with probability 1). By Markov's inequality, on all but at most $3p(N)$ fraction of x , this machine outputs $f(x)$ with probability at least $2/3$. Its space usage is

$$O(N^\delta) = O\left(n^{(k/\alpha)\delta}\right) = O(n^{ck\theta}) = O(n^\gamma).$$

Our assumption that $\text{prBP}^*\text{L} = \text{prL}$ then derandomizes the BP^*SPACE machine into a standard SPACE machine using the same amount of space. This proves Item (1). $\square$

As pointed out by Valentine Kabanets, by plugging in the unconditional algorithm that simulates $\text{TIME}[T]$ in space $\tilde{O}(\sqrt{T})$ by Williams [Wil25], we indeed obtain the following unconditional hardness for low-space Avoid.

Corollary 5.15. *Fix a polynomial time bound $t(n) \geq n$. For every constant $\varepsilon \in (0, 1/4)$, there is a family $\mathcal{C} = \{C_x\}_{x \in \{0,1\}^*}$ of Avoid instances stretching $\lceil \log^* n \rceil$ bits to $\lceil t(n) \rceil$ bits, where $n = |x|$, and evaluable in space $O(t(n)^{1/2+\varepsilon})$ such that $\text{Avoid}_{\mathcal{C}}$ is not solvable in time $O(t(n))$ by a deterministic algorithm (on every input) for sufficiently large n .*

Proof. Let $\ell(n) = \lceil \log^* n \rceil$, $s(n) = \lceil t(n)^{1/2+\varepsilon} \rceil$, and $m(n) = \lceil t(n) \rceil$. Following the universal-machine construction in Definition 5.13, define $C_x(d)$ by interpreting $d \in \{0,1\}^{\ell(n)}$ as a machine description and simulating it on x , with space bound $s(n)$ and time bound $2^{s(n)}$. The output is the simulated machine's $m(n)$ -bit output, or $0^{m(n)}$ if either bound is exceeded or the output length is incorrect. Each output bit can be evaluated in space $O(s(n))$.

Suppose that a deterministic algorithm A solves $\text{Avoid}_{\mathcal{C}}$ in time $O(t(n))$. Applying Williams' simulation [Wil25] separately to each output bit yields a machine M that prints $A(x)$ using space $\tilde{O}(\sqrt{t(n)}) = o(s(n))$ and time $2^{o(s(n))}$. Thus, for every sufficiently large n , the description $d_M \in \{0,1\}^{\ell(n)}$ of M satisfies $C_x(d_M) = A(x)$ for every $x \in \{0,1\}^n$, contradicting that $A(x)$ lies outside $\text{Range}(C_x)$. $\square$

6 The Complexity of Low-Space Avoid

6.1 Worst-case hardness for non-deterministic algorithms

Theorem 6.1 (low-space Avoid is hard in the worst-case). *If $\text{NP} \neq \text{PSPACE}$ then for every $\varepsilon > 0$ there is a family $\mathcal{C} = \{C_x : [4] \rightarrow \{0,1\}^{|x|^{\sqrt{\varepsilon}}}\}$ evaluable in space $|x|^\varepsilon$ such that $\text{Avoid}_{\mathcal{C}}$ is not solvable in non-deterministic polynomial time (in the worst-case).*

Proof. We will prove the contrapositive. Assuming that for some $\varepsilon > 0$ and every $\mathcal{C} = \{C_x\}$ as above there is a non-deterministic algorithm for $\text{Avoid}_{\mathcal{C}}$, we construct an NP verifier V for TQBF.

A suitable version of $\text{IP} = \text{PSPACE}$. We need a version of the protocol in which the arithmetization supports the following properties:

1. The field is a Healy-Viola field [HV06], i.e. of size $2^{2 \cdot 3^\ell}$, and in particular of characteristic two. This is so that we can perform arithmetic in space $\text{polyloglog}(|\mathbb{F}|)$.²⁶
2. The prover can output the coefficients of f_i in space $\text{poly}(|\varphi|, \log\log(|\mathbb{F}|))$. (We will be more precise below about the read-only input available to the prover.)
3. The degree bound d is constant, and in particular we will shoot for *degree three*. This is so that we can factor $f_i - p_i$ in space $\text{polyloglog}(|\mathbb{F}|)$ (e.g., without storing intermediate field elements in the factorization process).

Working over Healy-Viola fields calls for arithmetizing φ over characteristic two. Arithmetizing the 3CNF part is standard,²⁷ arithmetizing a $\forall x_i p_i(x_i)$ quantifier is just multiplication $\prod_{x_i \in \{0,1\}} p_i(x_i)$, and arithmetizing an $\exists x_i p_i(x_i)$ quantifier reduces by De-Morgan to multiplication $1 + \prod_{x_i \in \{0,1\}} (1 + p_i(x_i))$. (And the linearization operators are standard.)

To get an individual degree bound of $d = 3$, it suffices that each p_i has degree 3 in the corresponding variable x_i , and in turn it suffices that each variable x_i appears in at most three clauses. This follows by preprocessing: when given φ , the NP-machine creates an equivalent φ' in which for each variable x_i appearing in k clauses, each occurrence $j \in [k]$ is replaced with a new variable $x_{i,j}$, and there is another set of clauses enforcing consistency $(\neg x_{i,1} \vee x_{i,2}) \wedge (\neg x_{i,2} \vee x_{i,3}) \wedge \dots \wedge (\neg x_{i,k} \vee x_{i,1})$.

Finally, we want to show that the prover is computable in space $\text{poly}(n, \log\log(|\mathbb{F}|))$. Specifically, the Avoid instance C_{x_i} will be indexed by x_i that includes the (preprocessed) formula φ , the previous verifier challenges $r_1, \dots, r_i$, the polynomials $p_1, \dots, p_i$, and polynomially long padding, and we want to show that it can print the roots of $f_i - p_i$ in space $\text{poly}(n, \log\log(|\mathbb{F}|))$.

We first argue that, given $a \in \mathbb{F}$, we can compute $f_i(a)$ in such space. Recall that

$$f_i(a) = \hat{Q}_{i+1} \hat{Q}_{i+2} \dots \hat{\varphi}(r_1, r_2, \dots, r_{i-1}, a),$$

where the $\hat{Q}$'s are operators on polynomials (defined above) and $\hat{\varphi}$ is the arithmetization of φ . We claim that it suffices to show:

²⁶Actually, for reasons that will be explained below, we will work over a specific degree-two extension of a Healy-Viola field. This will maintain the crucial property for us, which is that arithmetic can be performed in space $\text{polyloglog}(|\mathbb{F}|)$.

²⁷Arithmetize $x_1 \vee \neg x_2 \vee x_3$ as $1 + (1 + x_1)(x_2)(1 + x_3)$, and multiply the arithmetizations of the clauses.

(*) Given input $(\hat{\varphi}, r_1, \dots, r_{i-1}, a, b_1, \dots, b_j)$ and oracle access to a function that takes $b \in \{0, 1\} \subseteq \mathbb{F}$ and outputs $\hat{Q}_{i+1}\hat{Q}_{i+2}\dots\hat{\varphi}(r_1, \dots, r_{i-1}, a, b_1, \dots, b_j, b)$, we can compute $\hat{Q}_{i+j+1}\hat{Q}_{i+j+2}\dots\hat{\varphi}(r_1, \dots, r_{i-1}, a, b_1, \dots, b_j)$ in space $\text{poly}(n, \log\log(|\mathbb{F}|))$.

Establishing (*) is enough to prove that $f_i(a)$ is computable in low space, because $f_i(a)$ is the base case where there are no b_j 's (i.e., $j = 0$), and we can unravel the oracle access at each layer of this recursion using emulative composition. There are $\text{poly}(n)$ levels of recursion, and at each level we only need to store $\text{poly}(n, \log\log(|\mathbb{F}|))$ bits (recall that the b_j 's can be represented by a single bit). And indeed, the claim (*) itself follows since we can easily decide which operator is $\hat{Q}_{i+j}$ from the number $i + j$ (i.e., in time $\text{poly}(n)$), considering a case-analysis for evaluation, and using the fact that arithmetic in a Healy-Viola field can be performed in space $\text{polyloglog}(|\mathbb{F}|)$.

The next step is interpolation: we want to show that we can output the coefficients of $f_i(a)$ in space $\text{poly}(n, \log\log(|\mathbb{F}|))$. This follows using the Lagrange interpolation: by inspection (after collating terms), each of the four coefficients of f_i can be computed from evaluations of f_i at four distinct points, using a constant number of arithmetic operations over $\mathbb{F}$.²⁸ Treating this sequence of arithmetic operations as a recursion with constantly many levels, and using emulative composition, we can compute the coefficients in space $\text{poly}(n, \log\log(|\mathbb{F}|))$.

The last step is factoring $f_i - p_i$. We run the algorithm of Kalai, Lombardi, and Vaikuntanathan [KLV23, Section 3] for factoring cubics, and provide it virtual access to $f_i - p_i$.²⁹ Their algorithm is a highly uniform TC^0 procedure over the input, which is of length $O(\log(|\mathbb{F}|))$, and simulating TC^0 in the usual space-bounded way yields the final space-bounded algorithm.

The Avoid family. Let $\varphi \in \{0, 1\}^n$ be a (description of) a totally quantified formula, and assume that it is already preprocessed as described above (i.e., every variable appears in at most three clauses). Let $\delta = \delta(\varepsilon) > 0$ be sufficiently small whose precise value will be spelled out below.

We consider the aforementioned degree-two extension of a Healy-Viola field $\mathbb{F}$ of size $O(2^{n^{1/\delta}})$, and an instantiation of the protocol above over this field. Let $R = \text{poly}(n)$ be the number of rounds, and denote verifier challenges by $r_1, \dots, r_R \in \mathbb{F}$ and prover responses (which are degree-3 univariates) by $p_1, \dots, p_R$.

We define the following family $\{C_x\}$: Parse $x = (\hat{\varphi}, r_1, \dots, r_i, p_1, \dots, p_i, 1^P)$ as a padded partial transcript; given an input $j \in [4]$, output the j^{th} root of $f_i - p_i$. Indeed, in case $f_i \equiv p_i$ this will just be the j^{th} element in the field.

²⁸Let $x_0, \dots, x_3 \in \mathbb{F}$ be the evaluation points, let $y_0, \dots, y_3 \in \mathbb{F}$ be the corresponding values, and let $d_i = \prod_{j \neq i} (x_i - x_j)$. Then:

$$\begin{aligned} p(X) &= \sum_{i=0}^3 y_i \frac{1}{d_i} \prod_{j \neq i} (X - x_j) \\ &= \frac{y_0}{d_0} \cdot (X^3 + (x_1 + x_2 + x_3)X^2 + (x_1x_2 + x_1x_3 + x_2x_3)X + x_1x_2x_3) \\ &\quad + \frac{y_1}{d_1} \cdot (X^3 + (x_0 + x_2 + x_3)X^2 + (x_0x_2 + x_0x_3 + x_2x_3)X + x_0x_2x_3) \\ &\quad + \frac{y_2}{d_2} \cdot (X^3 + (x_1 + x_0 + x_3)X^2 + (x_1x_0 + x_1x_3 + x_0x_3)X + x_1x_0x_3) \\ &\quad + \frac{y_3}{d_3} \cdot (X^3 + (x_1 + x_2 + x_0)X^2 + (x_1x_2 + x_1x_0 + x_2x_0)X + x_1x_2x_0) \end{aligned}$$

and by inspection, the coefficient of each X^k can be computed by constantly many operations on $x_0, \dots, x_3, y_0, \dots, y_3$.

²⁹There are two non-obvious points involved. The first is that we need to compute coefficients of $f_i - p_i$ from coefficients of f_i and of p_i without storing a field element; this is easy, since we just subtract the two relevant coefficients. The second point is that the [KLV23] algorithm is actually over a specific degree-two extension of a Healy-Viola field (using the irreducible $p(y) = y^2 + y + x^3$, where x^3 is an element in the base field). Thus, our arithmetization above is over this degree-two extension, and as they prove the extension still supports arithmetic operations in space $\text{polyloglog}(|\mathbb{F}|)$ (see [KLV23, Theorem 3.4]).

By the above, $C_x: [4] \rightarrow \{0, 1\}^{n^{1/\delta} + O(1)}$ is computable in space $\text{poly}(n, \log \log(|\mathbb{F}|)) \leq n^c$ for some universal constant $c > 1$. We choose δ and the padding length $P = n^C$ (which will dominate the length of x) such that $n^{1/\delta} \gg P^{\sqrt{\varepsilon}}$ (i.e., the output length of C_x is more than $|x|^{\sqrt{\varepsilon}}$) and $n^c \ll \min\{P^\varepsilon, n^{\sqrt{\varepsilon}/\delta}\}$ (i.e., the space complexity is less than $|x|^\varepsilon$ and less than the output length $n^{1/\delta} + O(1)$ to the power $\sqrt{\varepsilon}$). Any choice of $C > c/\varepsilon$ and $\delta < \min\{\sqrt{\varepsilon}/c, 1/(C \cdot \sqrt{\varepsilon})\}$ suffices.

The NP verifier for TQBF. Given a totally quantified formula $\varphi \in \{0, 1\}^n$ for TQBF, we preprocess it and arithmetize it as described above. The verifier V will make all non-deterministic guesses in advance, but it is instructive to describe it as making guesses on-the-fly. Specifically, for $i \in [R]$, the verifier:

1. Non-deterministically guesses $p_i: \mathbb{F} \rightarrow \mathbb{F}$ of degree 3. (We stress again that this guess can be made in advance.)
2. Checks that $v_{i-1} = \widehat{Q}_{i-1}p_i$, where $v_0 \triangleq 1$ and we abuse notation to have $\widehat{Q}_{i-1}$ act on univariates (e.g. $\widehat{Q}_{i-1}p_i = 1 + (1 + p_i(0)) \cdot (1 + p_i(1))$).
3. Constructs a partial transcript $x_i = (\varphi, r_1, \dots, r_{i-1}, p_1, \dots, p_i)$, and runs the Avoid algorithm on input x_i to obtain $r_i \in \mathbb{F}$, aborting if the algorithm outputs $\perp$.
4. Defines $v_i = p_i(r_i)$.

The final consistency check is the standard one, i.e. checking that $v_R = \widehat{\varphi}(r_1, \dots, r_R)$.

Completeness is straightforward, since when sending $p_i = f_i$ at each stage, any output r_i of the Avoid algorithm (i.e., any field element) will yield $v_i = p_i(r_i) = f_i(r_i)$. For soundness, the key point is that if $\varphi \notin \text{TQBF}$, then for the first consistency check to pass we must have $p_1 \neq f_1$. In this case the Avoid algorithm outputs r_1 on which f_1 and p_1 disagree, hence $v_1 \neq f_1(r_1)$. Continuing by induction, r_R will be such that $\widehat{\varphi}$ and p_R disagree on it, hence the final consistency check will fail. $\square$

6.2 Average-case hardness for tiny-error non-deterministic algorithms

In this section we prove Theorem 1.8. That is, assuming demi-bit generators against AM adversaries, we show a distribution of Avoid instances that is hard on average against non-deterministic algorithms that may err on a tiny fraction of inputs. The proof follows the description in Section 2.4: We first present an alternative proof of Ilango's [Il25], and in particular prove the covering lemma (i.e., Lemma 2.2); and then, building on that, we extend the result to algorithms that err on a small number of inputs. We note that the hard Avoid distribution is the same one given in [Il25], and only the analysis is different.

Reminder: Demi-bits. Our results will rely on the existence of a demi-bits generator, so we recall the standard definition and present a relaxed one that we will use.

Definition 6.2 (Demi-bits Generator). *We say that $g: \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ is an ε -demi-bits generator against NP if no NP verifier V satisfies the following:*

1. For at least $\varepsilon \cdot 2^n$ strings y it holds that $\exists w: V(y, w) = 1$.
2. For every y in the range of g it holds that $\forall w: V(y, w) = 0$.

Definition 6.3 (Relaxed Demi-bits against AM). *We say that $g: \{0, 1\}^\ell \rightarrow \{0, 1\}^n$ is a relaxed ε -demi-bits generator against AM if no AM verifier V satisfies the following:*

1. For at least $\varepsilon \cdot 2^n$ strings y it holds that $\Pr_r[\exists w: V(y, r, w) = 1] \geq 2/3$.
2. For every y in the range of g it holds that $\Pr_r[\exists w: V(y, r, w) = 1] < 1/3$.

The reason we call the generator above “relaxed” is that the requirement is every AM verifier with acceptance probability $\varepsilon > 0$ does not reject one of the pseudorandom strings (i.e. has acceptance probability more than $1/3$). The natural requirement is stricter, demanding that the verifier *accepts* a pseudorandom string (i.e., has acceptance probability at least $2/3$), yet the relaxed object suffices for our proof.

6.2.1 An alternative proof of the original result

We start by showing a combinatorial lemma that will be useful for us. This lemma is an abstraction and generalization of the main technical idea in [Ila25, Section 1.3], albeit with a different presentation and proof.

Consider an arbitrary dense set of tuples $S \subseteq \{0, 1\}^{t \times n}$, where each element in S takes the form of $\mathbf{s} = (s_1, \dots, s_t)$ for $s_i \in \{0, 1\}^n$. Next, consider an adversary that for every $\mathbf{s} \in S$ applies a shift $h(\mathbf{s})$ to all t strings to obtain a shifted tuple $\tilde{\mathbf{s}} = (s_1 \oplus h(\mathbf{s}), \dots, s_t \oplus h(\mathbf{s}))$. The goal of the adversary is to shift all tuples so that they “concentrate” on a small set in $\{0, 1\}^n$; that is, the number of n -bit strings that are elements of shifted tuples is small. In other words, the adversary tries to minimize the set

$$\{y \in \{0, 1\}^n \mid \exists i, \mathbf{s} \in S : s_i \oplus h(\mathbf{s}) = y\}.$$

The lemma below says that this is impossible.

Lemma 6.4 (covering lemma). *Let $n, t \in \mathbb{Z}$. Let $S \subseteq \{0, 1\}^{t \times n}$ be an arbitrary set of tuples such that $|S| \geq \theta \cdot 2^n$. Let $h : S \rightarrow \{0, 1\}^n$ be an arbitrary function. For any tuple $\mathbf{s} = (s_1, \dots, s_t) \in S$, we define the shifted tuple $\tilde{\mathbf{s}} := (h(\mathbf{s}) \oplus s_1, \dots, h(\mathbf{s}) \oplus s_t)$.*

*We call a string $y \in \{0, 1\}^n$ **covered** by a tuple $\mathbf{s}$ if it shows up in the shifted tuple $\tilde{\mathbf{s}} := (h(\mathbf{s}) \oplus s_1, \dots, h(\mathbf{s}) \oplus s_t)$. Define the set of strings G_h covered (at least once) by tuples in S as*

$$G_h = \{y \in \{0, 1\}^n \mid \exists i, \mathbf{s} : s_i \oplus h(\mathbf{s}) = y\}.$$

Then for any h it holds that $|G_h| \geq \theta^{1/t}$.

Proof. Consider the multi-set of all shifted tuples, while counting multiplicities (i.e., if k tuples become equal after they are shifted, we count that shifted tuple k times). We count the size of this set in two ways.

On the one hand, there are exactly $|S|$ tuples $\mathbf{s} \in S$ and hence exactly $|S|$ shifted tuples $\tilde{\mathbf{s}}$, counting multiplicities (i.e., each of the $|S|$ original tuples contributes exactly one shifted tuple).

On the other hand, by definition of a covered string, for any shifted tuple $\tilde{\mathbf{s}} = (y_1, \dots, y_t)$, all of y_i must be contained in G_h . Therefore the total number of distinct shifted tuples is at most $|G_h|^t$. Moreover, each distinct shifted tuple can have multiplicity at most 2^n , because $\tilde{\mathbf{s}} = \tilde{\mathbf{s}}'$ only if $\mathbf{s} \in \{\mathbf{s}' \oplus \mathbf{z} : \mathbf{z} \in \{0, 1\}^n\}$. Hence, the total number of shifted tuples is upper bounded by $|G_h|^t \cdot 2^n$.

Putting the two counts together we have

$$\theta \cdot 2^n \leq |S| \leq |G_h|^t \cdot 2^n,$$

and hence $|G_h| \geq \theta^{1/t}$. □

Using Lemma 6.4, we now show an alternative proof for Ilango’s [Ila25] original result.

Theorem 6.5. [Ila25, Theorem 3.3, Corollary 3.7] *For $t \in \mathbb{N}$, let $g : \{0, 1\}^{n-\lambda(n)} \rightarrow \{0, 1\}^n$ be an ε -demi-bits generator, where $\lambda(n) \geq \lceil \log t \rceil + 1$ ³⁰. For a tuple of strings $\mathbf{s} = (s_1, \dots, s_t) \in \{0, 1\}^{t \times n}$, define the circuit*

$$C_{\mathbf{s}}(x, i) := g(x) \oplus s_i.$$

There is no non-deterministic polynomial-time machine A that solves Avoid on $C_{\mathbf{s}}$ over uniform choices of $\mathbf{s}$ with the following parameters:

- *Completeness:* $\Pr_{\mathbf{s}}[\exists w : A(C_{\mathbf{s}}, w) \in \{0, 1\}^n \setminus \text{Range}(C_{\mathbf{s}})] \geq 2^n \cdot \varepsilon^t$.
- *Soundness:* $\Pr_{\mathbf{s}}[\forall w : A(C_{\mathbf{s}}, w) \notin \text{Range}(C_{\mathbf{s}})] = 1$. (Note that A is allowed to output $\perp$.)

Proof. Let D be the following non-deterministic machine:

On input $y \in \{0, 1\}^n$, accept if and only if $\exists \mathbf{s}, w, i$ such that $A(C_{\mathbf{s}}, w) = y \oplus s_i$.

³⁰This is to ensure that the circuit $C_{\mathbf{s}}$ taking an additional $\lceil \log t \rceil$ -bit input remains a valid Avoid instance.

Soundness. Suppose $y = g(x)$ is in the image of g . By definition, for every $\mathbf{s}$ we have $C_{\mathbf{s}}(x, i) = g(x) \oplus s_i$, and hence $y \oplus s_i$ is in the image of $C_{\mathbf{s}}$. By the soundness of A , we deduce that $\forall w, A(C_{\mathbf{s}}, w) \neq y \oplus s_i$. Therefore, D does not accept y .

Completeness. It remains to show that D accepts at least $\varepsilon \cdot 2^n$ strings in $\{0, 1\}^n$. Let $G \subseteq \{0, 1\}^n$ be the set of strings accepted by D . Our goal is to show that $|G| \geq \varepsilon \cdot 2^n$.

Define the set S as choices of $\mathbf{s}$ where $A(C_{\mathbf{s}}, w)$ gives a non- $\perp$ output on some witness w .

$$S = \{\mathbf{s} \in \{0, 1\}^{t \times n} \mid \exists w : A(C_{\mathbf{s}}, w) \neq \perp\}$$

By the completeness of A , we have $|S| \geq \varepsilon^t \cdot 2^n \cdot 2^{tn}$. Also, by definition, $y \in \{0, 1\}^n$ is in G if $\exists \mathbf{s} \in S, w, i$ such that $A(C_{\mathbf{s}}, w) = y \oplus s_i$. In this case, we say that y is covered by $\mathbf{s}$ with shift value $A(C_{\mathbf{s}}, w)$. Our goal now reduces to showing many y 's are covered by some tuples with respect to shifts specified by A .

Towards that end, define $h : S \rightarrow \{0, 1\}^n$ such that $h(\mathbf{s}) = z$ if $\exists w, A(C_{\mathbf{s}}, w) = z$. (Pick one arbitrarily if there are multiple z 's that satisfy this. Indeed, we are restricting $A(C_{\mathbf{s}})$ to behave as a function, which can only reduce the number of covered strings.) By applying Lemma 6.4 with S and h and $\theta = (\varepsilon 2^n)^t$, it follows that $|G| \geq |G_h| \geq \varepsilon \cdot 2^n$, as needed. $\square$

6.2.2 Proof of Theorem 1.8

We now show that Avoid is hard against non-deterministic algorithms that makes a tiny fraction of error (instead of zero-error) over the input distribution, assuming demi-bit generators secure against AM adversaries. We comment that if the demi-bit generator can be evaluated in low space, then the hard indexed Avoid family can be evaluated in low space (as required in Assumption 1.1). We first prove a general technical statement, and then deduce Theorem 1.8 as a corollary.

Theorem 6.6 (hardness of Avoid against non-deterministic algorithms that err). *Let $g : \{0, 1\}^{n-\lambda(n)} \rightarrow \{0, 1\}^n$ be a relaxed ε -demi-bits generator against AM. Let $t = t(n)$ be polynomially bounded and let $\kappa = \kappa(n) \in (0, 1)$. Assume that $\lambda(n) \geq \lceil \log t(n) \rceil + 1$. For a tuple of strings $\mathbf{s} = (s_1, \dots, s_t) \in \{0, 1\}^{t \times n}$, define the circuit*

$$C_{\mathbf{s}}(x, i) := g(x) \oplus s_i.$$

Here, i is represented using $\lceil \log t \rceil$ bits; if t is not a power of two, map every unused encoding of i to s_1 . There is no non-deterministic polynomial-time machine A that solves Avoid on $C_{\mathbf{s}}$ over uniform choices of $\mathbf{s}$ with the following parameters:

- *Completeness:* $\Pr_{\mathbf{s}}[\exists w : A(C_{\mathbf{s}}, w) \in \{0, 1\}^n \setminus \text{Range}(C_{\mathbf{s}})] \geq 2^n \cdot 2\kappa + 2^n \cdot \varepsilon^t$.
- *Soundness:* $\Pr_{\mathbf{s}}[\forall w : A(C_{\mathbf{s}}, w) \notin \text{Range}(C_{\mathbf{s}})] \geq 1 - \kappa$. (Note that A is allowed to output $\perp$.)

Recall in the proof of Theorem 6.5 above, we showed that a large fraction of strings are covered by (at least one of the) tuples; and none of the strings in the image of g are covered by any tuple, due to zero-error property of the assumed algorithm A . This NP-property (i.e., “is there a covering tuple”) allowed us to distinguish between the set of pseudorandom strings and a large set of truly random strings.

However, once we relax A and allows it to make error, the soundness property that none of the strings in the image of g are covered is no longer true. Indeed, if A makes error on κ fraction of choices of $\mathbf{s}$, then a pseudorandom string could potentially be covered by $\kappa \cdot 2^{tn}$ tuples (i.e. tuples $\mathbf{s}$ that A errs on).

Nonetheless, if κ is sufficiently small, one can hope to show that many random strings are covered by $2\kappa \cdot 2^{tn}$ tuples. Indeed, this is precisely what we show, by extending the analysis in the proof of Lemma 6.4. And at that point one can apply the Goldwasser-Sipser Set Size Lower Bound protocol in AM to distinguish these two cases (i.e., at most κ fraction of tuples vs at least 2κ fraction of tuples). This is why we require the demi-bit to be secure against AM adversaries.

Proof of Theorem 6.6. Assuming towards a contradiction that such A exists, we design an AM protocol that contradicts the security of g .

Define a non-deterministic verifier V on input $y \in \{0, 1\}^n$, $\mathbf{s} = (s_1, \dots, s_t) \in \{0, 1\}^{t \times n}$ as follows:

$$V(y, \mathbf{s}) = 1 \iff \exists w, i : A(C_{\mathbf{s}}, w) = y \oplus s_i.$$

For any fixed y , define the set of accepting $\mathbf{s}$ as $T_y \triangleq \{\mathbf{s} \in \{0, 1\}^{t \times n} \mid V(y, \mathbf{s}) = 1\}$. I.e. these are tuples that cover y with respect to the shift relation specified by A .

We will show the followings:

1. (Completeness): For at least $\varepsilon \cdot 2^n$ strings y in $\{0, 1\}^n$, $|T_y| \geq 2\kappa \cdot 2^{tn}$.
2. (Soundness): For every $y \in \text{Range}(g)$, $|T_y| \leq \kappa \cdot 2^{tn}$.

It then follows from the Goldwasser-Sipser Set Size Lower Bound protocol (Theorem 3.9) and standard constant-error amplification that there is an AM protocol accepting with probability at least $2/3$ on at least $\varepsilon \cdot 2^n$ strings y and with probability strictly less than $1/3$ on every output of g . We now proceed to prove the two items.

Soundness. Suppose $y = g(x)$ is in the image of g . By definition of $C_{\mathbf{s}}$, for all choices of $\mathbf{s}$ and all $i \in [t]$, we know that $y \oplus s_i = C_{\mathbf{s}}(x, i)$ is in the image of $C_{\mathbf{s}}$. By the soundness of A , for every $y \in \text{Range}(g)$,

$$\Pr_{\mathbf{s}}[V(y, \mathbf{s}) = 1] \leq \Pr_{\mathbf{s}}[\exists w : A(C_{\mathbf{s}}, w) \in \text{Range}(C_{\mathbf{s}})] \leq \kappa.$$

Therefore, $|T_y| \leq \kappa \cdot 2^{tn}$.

Completeness. Let $G := \{y \in \{0, 1\}^n \mid |T_y| \geq 2\kappa \cdot 2^{tn}\}$ be the set of strings that are covered by at least a 2κ -fraction of tuples. Our goal is to show that $|G| \geq \varepsilon 2^n$.

To do this, we will first show that the set S of tuples that only cover elements in G is large, and then use Lemma 6.4 to show that these tuples must cover many elements (hence G is large). The fact that S is large will follow from the completeness of A (i.e., many tuples cover some elements), and since each element that is *not* in G is covered only by a few tuples. Details follow.

Let S_1 be the set of tuples $\mathbf{s}$ where $A(C_{\mathbf{s}}, w)$ gives a non- $\perp$ output on some witness w , i.e.

$$S_1 = \{\mathbf{s} \in \{0, 1\}^{t \times n} \mid \exists w : A(C_{\mathbf{s}}, w) \neq \perp\}$$

By the completeness of A , we have $|S_1| \geq (2\kappa \cdot 2^n + \varepsilon^t \cdot 2^n) \cdot 2^{tn}$. Now let $B := \{0, 1\}^n \setminus G$ (i.e., strings that covered by less than $2\kappa \cdot 2^{tn}$ tuples), and let $S_2 \subseteq S_1$ be the set of tuples $\mathbf{s}$ that cover any element in B :

$$S_2 = \{\mathbf{s} \in S_1 \mid \exists w, i : A(C_{\mathbf{s}}, w) \oplus s_i \in B\}$$

Finally, define $S = S_1 \setminus S_2$. By definition of S_2 , all strings covered by tuples in S must fall in G . Moreover $|S_2| \leq |B| \cdot 2\kappa \cdot 2^{tn} \leq 2^n \cdot 2\kappa \cdot 2^{tn}$ since each string in B can be covered by at most $2\kappa \cdot 2^{tn}$ tuples. Hence $|S| \geq \varepsilon^t \cdot 2^n \cdot 2^{tn}$.

Next, define the mapping $h : S \rightarrow \{0, 1\}^n$ such that $h(\mathbf{s}) = z$ if $\exists w, A(C_{\mathbf{s}}, w) = z$. (Pick one arbitrarily if there are multiple z 's that satisfies this.) By applying Lemma 6.4 with S and h , it follows that $|G| \geq |G_h| \geq (\varepsilon^t 2^{tn})^{1/t} = \varepsilon 2^n$, as needed. $\square$

Theorem 1.8 follows by a suitable choice of parameters.

Corollary 6.7 (Theorem 1.8, restated). *Let $g : \{0, 1\}^{n - \lceil \log n \rceil - 1} \rightarrow \{0, 1\}^n$ be a relaxed $1/4$ -demi-bits generator against AM. For a tuple $\mathbf{s} = (s_1, \dots, s_n) \in \{0, 1\}^{n \times n}$, define the circuit $C_{\mathbf{s}} : \{0, 1\}^{n-1} \rightarrow \{0, 1\}^n$ by*

$$C_{\mathbf{s}}(x, i) = g(x) \oplus s_i,$$

where, $x \in \{0, 1\}^{n - \lceil \log n \rceil - 1}$ and i is represented using $\lceil \log n \rceil$ bits, with unused encodings mapped to s_1 .

Then, there is no non-deterministic polynomial-time machine A that for every sufficiently large n satisfies

$$\Pr_{\mathbf{s} \in \{0,1\}^{n \times n}} [\exists w : A(C_{\mathbf{s}}, w) \in \{0, 1\}^n \setminus \text{Range}(C_{\mathbf{s}})] \geq 3 \cdot 2^{-n}$$

and

$$\Pr_{\mathbf{s} \in \{0,1\}^{n \times n}} [\forall w : A(C_{\mathbf{s}}, w) \notin \text{Range}(C_{\mathbf{s}})] \geq 1 - 2^{-2n}.$$

Observe that the complexity of evaluating $C_{\mathbf{s}}$ boils down to the complexity of computing g (i.e., it only requires additional auxiliary $O(\log n)$ space on top of computing g).

Proof. Apply Theorem 6.6 with demi-bits parameter $1/4$, $\kappa(n) = 2^{-2n}$, $t(n) = n$, and $\lambda(n) = \lceil \log n \rceil + 1$. The circuit $C_{\mathbf{s}}$ has $n - 1$ input bits, and

$$2^n (2\kappa + (1/4)^t) = 2^{-n+1} + 2^{-n} = 3 \cdot 2^{-n},$$

for all sufficiently large n , while $1 - \kappa = 1 - 2^{-2n}$. Thus, any machine satisfying the two displayed conditions would contradict Theorem 6.6. $\square$

Remark 6.8. The dependence of the completeness threshold in Theorem 6.6 on κ is essentially optimal for the above covering argument, up to a polynomial factor in t . Indeed, let S be the set of tuples on which A succeeds, and fix one valid non- $\perp$ output for each tuple in S . Each resulting shifted tuple covers at most t strings. Therefore, for at least $\varepsilon \cdot 2^n$ strings to each be covered by at least $2\kappa \cdot 2^{tn}$ tuples, incidence counting requires

$$t \cdot |S| \geq \varepsilon \cdot 2^n \cdot 2\kappa \cdot 2^{tn}.$$

Equivalently, the success probability of A must be at least

$$\frac{2\varepsilon}{t} \cdot 2^n \kappa.$$

Thus, when ε is constant and t is polynomially bounded, the $2^n \kappa$ dependence in Theorem 6.6 is optimal for this argument up to a polynomial factor.

A covering threshold larger than κ is also necessary. The κ -fraction of tuples on which A may err could all be aligned with one fixed string $y \in \text{Range}(g)$: on each such tuple $\mathbf{s}$, an erroneous witness may cause A to output $y \oplus s_i \in \text{Range}(C_{\mathbf{s}})$ for some i . Consequently, a string in $\text{Range}(g)$ may be covered by a κ -fraction of the tuples. The factor 2 provides the constant-factor gap needed by the set-size protocol.

6.3 An average-case non-deterministic algorithm

In this section we prove Theorem 1.6. Since we will prove the contrapositive, let us state the converse of its conclusion as an assumption, and see that this assumption contradicts the existence of sub-exponentially secure one-way functions.

Theorem 6.9 (Theorem 1.6, restated). *If there are sub-exponentially secure one-way functions, then for some $\varepsilon > 0$ there is a non-deterministic polynomial-time algorithm A that uses only n^ε bits of non-determinism and satisfies the following. For every family of Avoid instances $C = \{C_x\}$ stretching $|x|^{\varepsilon^2}$ bits to $|x|^\varepsilon$ bits and evaluable in time $2^{|x|^{\varepsilon^2}}$, and infinitely many $n \in \mathbb{N}$,*

$$\Pr_{x \in \{0,1\}^n} [A(x) \text{ solves Avoid}(C_x)] \geq 1/\text{poly}(n),$$

where the meaning of “solves Avoid(C_x)” is that on some witness $A(x)$ outputs an Avoid solution, and for every witness it outputs either an Avoid solution or $\perp$.

Proof. The works of Liu and Pass [LP20, LP22] show a problem $L^{\text{hard}} \in \text{NP}$ such that if L^{hard} can be solved with probability $1 - n^{-\omega(1)}$ of success over a uniform choice of input for every $n \in \mathbb{N}$, then there are no one-way functions. Their reduction (of breaking a one-way function to solving L^{hard}) is a polynomial-time algorithm that uses the algorithm for L^{hard} as a black-box, and only blows up the input length polynomially. Hence, for every $\varepsilon > 0$, if the algorithm for L^{hard} runs in time 2^{n^γ} for a sufficiently small $\gamma = \gamma(\varepsilon) > 0$, we obtain an algorithm running in time 2^{n^ε} for breaking any one-way function.

Now, instantiate [Theorem 5.4](#) with this $L = L^{\text{hard}}$, with $\mathcal{XY}$ being the uniform distribution, and with $p(N) = N^{-\omega(1)}$. Recall (by [Remark 5.5](#)) that there is a non-deterministic polynomial-time algorithm $A = A_L$ such that if A fails to solve Avoid for some $C = \{C_x\}$ (with parameters as in our statement) then there is an AM verifier V that decides L correctly on at least $1 - p(N) = 1 - n^{-\omega(1)}$ of the n -bit inputs such that V receives $T^{\delta/\varepsilon}$ bits from the prover. To decide L we simulate V (using random coins), enumerate over all possible prover messages, whose total number is $2^{T^{\delta/\varepsilon}} = 2^{n^{k\delta/\varepsilon}}$, and accept if and only if there is a prover message that convinces V . Since $\delta = \varepsilon^2$ and this is true for an arbitrarily small $\varepsilon > 0$, we obtain an algorithm that breaks any one-way function in time 2^{n^γ} for an arbitrarily small $\gamma > 0$, a contradiction. $\square$

Next, by assuming the existence of sub-exponentially secure one-way permutations, we show that there is a non-deterministic polynomial-time algorithm that solves low-space Avoid with high probability over a uniform choice of input, strengthening the result above.

Definition 6.10 (sub-exponentially secure strong one-way permutation). *A sub-exponentially secure strong one-way permutation is a polynomial-time computable permutation $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$, such that for some constant $\rho \in (0, 1)$, every uniform probabilistic inverter I running in time at most 2^{n^ρ} satisfies*

$$\Pr_{u \in \{0, 1\}^n, I} [I(1^n, u) = f_n^{-1}(u)] \leq n^{-\omega(1)}$$

for all sufficiently large n .

Theorem 6.11 (a high-success non-deterministic algorithm from one-way permutations). *If there are sub-exponentially secure strong one-way permutations, then for some $\varepsilon > 0$ there is a non-deterministic polynomial-time algorithm A that uses only n^ε bits of non-determinism and satisfies the following. For every family of Avoid instances $C = \{C_x\}$ stretching $|x|^{\varepsilon^2}$ bits to $|x|^\varepsilon$ bits and evaluable in time $2^{|x|^{\varepsilon^2}}$, and infinitely many $n \in \mathbb{N}$,*

$$\Pr_{x \in \{0, 1\}^n} [A(x) \text{ solves Avoid}(C_x)] \geq 1 - n^{-\omega(1)},$$

where the meaning of “solves Avoid(C_x)” is as in [Theorem 6.9](#).

Proof. Suppose towards a contradiction that an algorithm as in the conclusion does not exist. Fix a one-way permutation f with security exponent ρ as in [Definition 6.10](#), and choose a constant $0 < \varepsilon < \rho$.

Consider the following algorithm A for Avoid (indeed, we are setting A up to fail). On input $x = u \circ r \in \{0, 1\}^N$, where $|u| = m(N) = N^\varepsilon$, the algorithm $A(x)$ guesses $w \in \{0, 1\}^{m(N)}$, outputs w if $f_{m(N)}(w) = u$, and otherwise outputs $\perp$. This is a non-deterministic polynomial-time algorithm using at most N^ε non-deterministic bits. Since f is a permutation, A has exactly one accepting path on every input. Thus, the assumption implies that there is some Avoid family $C = \{C_x\}$ (with parameters as in the statement) such that for every sufficiently large N ,

$$\Pr_{x = u \circ r \in \{0, 1\}^N} [f_{m(N)}^{-1}(u) \in \text{Range}(C_x)] \geq N^{-c}.$$

We now construct a uniform probabilistic inverter I for f . Given $(1^m, u)$ with $u \in \{0, 1\}^m$, set $N = m^{1/\varepsilon}$, and sample a uniform $r \in \{0, 1\}^{N-m}$. Enumerate all inputs z of $C_{u \circ r}$, compute $v = C_{u \circ r}(z)$, and return v if $f_m(v) = u$. If no input yields an inverse, output $\perp$. Whenever the inverse belongs to $\text{Range}(C_{u \circ r})$, this enumeration finds the inverse. Consequently, for every sufficiently large m ,

$$\Pr_{u \in \{0, 1\}^m, I} [I(1^m, u) = f_m^{-1}(u)] \geq N^{-c} = m^{-c/\varepsilon}.$$

There are $2^{N^{\varepsilon^2}}$ possible inputs z , and each output bit is evaluable in time $2^{N^{\varepsilon^2}}$. Therefore the inverter runs in time

$$\begin{aligned} T_I(m) &\leq \text{poly}(N) + 2^{N^{\varepsilon^2}} (m 2^{N^{\varepsilon^2}} + \text{poly}(N)) \\ &= 2^{O(N^{\varepsilon^2})} = 2^{O(m^{\varepsilon})} \leq 2^{m^{\rho}}, \end{aligned}$$

for all sufficiently large m . This contradicts the security of f . $\square$

7 Targeted Hitting-Set Generators for coNP (and Avoid)

As explained in [Section 1.4](#), a natural strategy for solving Avoid is to use a targeted hitting-set generator and to combine its outputs (to a single candidate solution). One situation in which combining outputs is feasible is when the number of outputs is small, i.e. sub-linear (see, e.g., [Section 7.1](#)).

We make progress on understanding the possibilities and limitations of such generators. To explain our results, let us recall the definition of targeted hitting-set generators by Goldreich [[Gol11a](#), [Gol11b](#)],³¹ and a main open problem that he posed and that is relevant in this context.

Definition 7.1 (targeted hitting-set generator). *Let $\mathcal{D}$ be a class of uniform probabilistic algorithms. We say that an algorithm G is a targeted hitting-set generator with error ε for $\mathcal{D}$ if for every $D \in \mathcal{D}$ and every $x \in \{0, 1\}^n$ (where n is sufficiently large) the following holds. If $\Pr_r[D(x, r) = 1] \geq \varepsilon(n)$, then the output-set of $G(x)$ contains a string w such that $D(x, w) = 1$.*

The definition extends naturally to D 's that are co-nondeterministic procedures, by defining $D(x, r) = 1$ if all non-deterministic choices cause $D(x, r)$ to accept and defining $D(x, r) = 0$ otherwise (i.e., if there is a choice causing D to reject). The definition also extends naturally to G that is computable non-deterministically; in this case, we require that for some witness $G(x)$ outputs a suitable set of strings, and for all witnesses $G(x)$ outputs either such a set or $\perp$.

Goldreich showed that the derandomization conjecture $\text{prBPP} = \text{prP}$ is equivalent to the existence of a polynomial-time computable targeted pseudorandom generator for uniform algorithms running in fixed polynomial time. A natural conjecture is that this result extends to derandomization of proof systems; that is, that $\text{prAM} = \text{prNP}$ is equivalent to the existence of a suitable targeted generator. Specifically, the suitable generator would be a targeted hitting-set generator computable in non-deterministic polynomial-time, and it would hit co-nondeterministic algorithms running in fixed polynomial time.

While such a generator suffices for $\text{prAM} = \text{prNP}$, Goldreich only showed that the latter conclusion implies a generator computable in polynomial time with an NP oracle, rather than in non-deterministic polynomial time. He posed obtaining a generator computable in non-deterministic polynomial time (which would yield a full equivalence to $\text{prAM} = \text{prNP}$) as an open problem.

Open Problem 1 (see [[Gol11a](#), Open Problems Section]). *Prove the following: If $\text{prAM} = \text{prNP}$, then there exists a targeted hitting-set generator with error $1/2$ that is computable in non-deterministic polynomial-time that hits co-nondeterministic algorithms running in linear time.*

Open Problem 1 has been studied in recent works [[vMS23a](#), [vMS23b](#)], as part of the recent interest in targeted generators (following the construction in [[CT21a](#)]; see [[CT23a](#)] for a survey).

Two new results. We focus on the special case in which the generator has a small output-set, of size $\ll |x|$. Prior works did not distinguish between this case and the general case in which the output set may be large (say, of polynomial size).³² This special case is of particular interest in the current paper (because it allows

³¹In the original paper only targeted pseudorandom generators are defined, but the extension to targeted hitting-set generators is straightforward.

³²For example, Goldreich's proof that $\text{prBPP} = \text{prP}$ is equivalent to efficient targeted pseudorandom generators for algorithms yields generators with a tiny output-set (depending on the precise definition, the pseudorandom set may be of size $O_\varepsilon(1)$ where ε is the required error, or of size $O_\varepsilon(f(n))$ where $f = \omega(1)$ is any growing function).

us to combine the outputs and obtain an Avoid solution), and it is also central in the study of superfast derandomization (i.e., derandomization with minimal overhead, which exploits the fact that the number of pseudorandom strings is small; see, e.g., [DMOZ22, CT21b, CT21a, SV22, CT23b, DT23, DPT24, BCT25, DMOZ26]). We prove two results concerning this special case.

1. In [Section 7.1](#) we prove that, assuming worst-case hardness of Avoid, there is no (non-deterministically computable) targeted hitting-set generator with a small output-set that fools coNP on every input x . The meaning is that, under the assumption, the answer to Goldreich’s question is *negative* in the special case of a generator with a small output-set.
2. In [Section 7.2](#) we show that, assuming sub-exponentially secure one-way functions, there is a non-deterministically computable targeted hitting-set generator with a small output-set that fools a natural subclass of coNP and works on average for a random input x .

7.1 There is no targeted generator with few outputs hitting coNP in the worst-case

The following result is based on the observation that a targeted hitting-set generator with a small output-set fooling coNP can be used to solve Avoid, by concatenating its outputs. We encourage the reader to parse the statement with output length $m(n) = n^\varepsilon$ and $k(n) = n^{O(\varepsilon)}$.

Theorem 7.2. *Let $m(n) < n$, and let $\mathcal{D}$ be the class of co-nondeterministic linear-time machines using $m(n)$ bits of randomness on inputs of length n . Assume that for every non-deterministic machine M there is $C: \{0, 1\}^{m - \lceil \log(k/m) \rceil - 1} \rightarrow \{0, 1\}^k$ of size n (for some $k = k(n) < n$ such that k/m is a power of two and $\log(k/m) + 1 < m$) such that M fails to solve Avoid on C .³³ Then there is no SearchNP -computable targeted $(1/2)$ -HSG against $\mathcal{D}$ that on n -bit inputs has seed length $\log(k/m)$ and output length m .*

Proof. Assume towards a contradiction that there exists such a generator, denoted by H . We use the following (failed) algorithm A for Avoid: Given an input circuit $C: \{0, 1\}^{m - \lceil \log(k/m) \rceil - 1} \rightarrow \{0, 1\}^k$ of size n , we non-deterministically compute $H(C)$ to obtain $r_1, \dots, r_L \in \{0, 1\}^m$ where $L = k/m$ (aborting if H outputs $\perp$), and output $y = r_1 \circ \dots \circ r_L$.

By our hardness assumption for Avoid, there is $C \in \{0, 1\}^n$ such that either $A(C)$ rejects all non-deterministic witnesses or on some witness $A(C)$ outputs $y = r_1 \circ \dots \circ r_L$ that is not an Avoid solution. Since for every C there is a witness causing $H(C)$ to output a list (because H is a valid targeted generator), the first case is impossible. In the second case, consider the linear-time co-nondeterministic machine D that uses m bits of randomness and, on input (C, r) , non-deterministically guesses z and $i \in [L]$ and rejects if the i th m -bit block of $C(z)$ equals r . Then D satisfies $\Pr_r[D(C, r) = 0] \leq 1/2$,³⁴ but rejects all of the r_i ’s. This contradicts the assumption that H hits $\mathcal{D}$. $\square$

[Theorem 7.2](#) referred to the standard formulation of Avoid. We also note that a similar conclusion holds for the low-space variant of Avoid studied in this work. We omit its proof for brevity as it follows from essentially the same proof as above.

Theorem 7.3. *Let $S(n) \geq \log n$, and let $m(n) \leq k(n) < n$, where k/m is a power of two and $\ell = m - \log(k/m) - 1 > 0$. Let $\mathcal{C}[\ell \rightarrow k, s]$ be a $\text{SPACE}(S(n))$ -evaluable indexed family such that $\text{Avoid}_{\mathcal{C}} \notin \text{SearchNP}$. Let $\mathcal{D}$ be the class of co-nondeterministic machines running in simultaneous time $2^{O(S(n))}$ and space $O(S(n) + \ell(n))$ and using $m(n)$ bits of randomness on inputs of length n . Then there is no SearchNP -computable targeted $(1/2)$ -HSG against $\mathcal{D}$ that on n -bit inputs has seed length $\log(k/m)$ and output length m .*

³³That is, either $M(C)$ rejects all witnesses, or on some witness $M(C)$ outputs a string in the range of C .

³⁴Indeed, $\Pr_{r \in \{0, 1\}^m}[D(C, r) = 0] = \Pr_r[\exists z, i : C(z)_i = r] \leq 2^{(m - \lceil \log(k/m) \rceil - 1) - m + \lceil \log(k/m) \rceil} = 1/2$.

7.2 A targeted generator with few outputs hitting a subclass of coAM on average

We now present a conditional construction of a targeted hitting-set generator with a small output-set that fools a subclass of coNP on average. The subclass consists of coNP procedures that use few (co-)non-deterministic bits. In fact, we prove a stronger result, in which the generator fools the subclass of coAM procedures (rather than just coNP procedures) that use few (co-)non-deterministic bits.

The main technical tool that we will use is a new targeted hitting-set generator with a small output-set, which is based on hardness of any problem computable in non-deterministic *sub-linear time*. This generator will combine the PCPs for sub-linear time from Section 4 with a known construction of a classical generator by Miltersen and Vinodchandran [MV05]. This will be shown in Section 7.2.2.

The analysis showing that this generator works uses a result of Liu and Pass [LP20, LP22] showing average-case hardness of an NP problem (assuming the existence of one-way functions). We will rely on a particular feature of their result, which is that their hard NP problem is actually computable in sub-linear time. The analysis appears in Section 7.2.3.

7.2.1 Definitions

For clarity, let us spell out the definition of a targeted hitting set generator against dense coAM procedures. For a coAM procedure P taking input x , randomness r and (co-)nondeterministic choice w , we will often suppress r and w in our notation, and denote $P(x)$; the meaning is then as follows:

Definition 7.4. *For a coAM procedure P taking input x , randomness r and witness w , we denote $P(x) = 1$ if $\Pr_r[\exists w : P(x, r, w) = 0] \leq 1/3$ and $P(x) = 0$ if $\Pr_r[\exists w : P(x, r, w) = 0] \geq 2/3$. We will sometimes consider coAM procedures that take input (x, y) rather than only x , in which case the notation $P(x, y)$ is defined identically.*

Definition 7.5 (targeted HSG against coAM procedures). *For $T, L, m, r, w : \mathbb{N} \rightarrow \mathbb{N}$ and $\varepsilon : \mathbb{N} \rightarrow [0, 1]$, let H be a non-deterministic algorithm that gets as input $x \in \{0, 1\}^n$ and, on every accepting path, outputs $L(n)$ strings of length $m(n)$. We say that H is a $(L, 1 - \varepsilon)$ -targeted hitting set generator against coAM time (T, m, r, w) if the following holds. For every coAM procedure D taking input $(x, y) \in \{0, 1\}^n \times \{0, 1\}^{m(n)}$, using $r(n)$ random bits and a $w(n)$ -bit witness, and running in time $T(n)$, for all sufficiently large $n \in \mathbb{N}$ and every $x \in \{0, 1\}^n$ such that $\Pr_{y \in \{0, 1\}^{m(n)}}[D(x, y) = 1] \geq 1 - \varepsilon(n)$, the following holds:*

1. $H(x)$ has at least one accepting path.
2. On any accepting path, let $s_1, \dots, s_{L(n)}$ be the output of $H(x)$. There exists $i \in [L(n)]$ such that $D(x, s_i) = 1$.

7.2.2 A targeted generator based on hard problems computable in sublinear time

We will make use of the following variant of the hitting set generator from [MV05], see also [GSTS03, Lemma 16]. (Note that the generator from [MV05] has super-linearly many output strings in the length of its input f . We will use it as a sub-component in our construction.)

Theorem 7.6. *Let $\gamma > 0$, and for a constant $d \in \mathbb{N}$ let $q(n)$ be the largest power of two such that $n \geq q^d \cdot \log(q)$, and let $m(n) = q(n) \cdot \log(q(n))$. There is an algorithm $MV = MV_{m, \gamma}$ and Arthur–Merlin verifiers V_c and V_e satisfying the following.*

- **Generator.** *Given an input $f \in \{0, 1\}^n$, the algorithm $MV(f)$ runs in time $\text{poly}(n)$ and outputs a list $S = (y_1, \dots, y_s)$ of m -bit strings of size $s \leq \text{poly}(n)$, which is intended to be a hitting set for coAM procedures.*
- **Reconstruction verifiers complexities:**

- **Commitment verifier.** The verifier V_c gets input n (in binary), sends $\tilde{O}(m^\gamma)$ bits of public randomness, expects $\tilde{O}(n^\gamma)$ bits from the prover, and outputs the randomness and the prover's message as a commitment string α .
- **Evaluation verifier.** The verifier V_e gets as input $i \in [n]$ and a commitment $\alpha \in \{0, 1\}^{\tilde{O}(n^\gamma)}$, and gets oracle access to a coAM procedure $D : \{0, 1\}^m \rightarrow \{0, 1\}$. Suppose that the underlying deterministic verifier $D(y, \rho, w)$ runs in time T_D , uses r_D random bits, and uses a w_D -bit witness. The verifier V_e sends a total number of $\tilde{O}(n^\gamma m r_D)$ bits of public randomness, receives a total number of $\tilde{O}(n^\gamma m(1 + w_D))$ bits from the prover,³⁵ runs in time

$$\text{poly}(m, n^\gamma),$$

and outputs a bit (or $\perp$).

More specifically, V_e receives $\tilde{O}(n^\gamma)$ strings $y \in \{0, 1\}^m$ for which the prover claims that $D(y) = 0$. For each such string, V_e uses $\Theta(m)$ independent random strings $\rho \in \{0, 1\}^{r_D}$ and receives a w_D -bit witness for each repetition. It accepts the claim $D(y) = 0$ only if, in a strict majority of the repetitions, the supplied witness causes $D(y, \rho, w)$ to reject. This deterministic evaluation of D is the only way in which V_e accesses its oracle.

- **Reconstruction verifiers correctness:**

- **Conformity.** If $D(y) = 0$ for every $y \in \text{MV}_{m, \gamma}(f)$, then there is a pair of prover strategies (P_c, P_e) such that for every $i \in [n]$,

$$\Pr[\langle P_e, V_e^D \rangle(i, \alpha) = f_i] \geq 1 - 2^{-\Omega(m)},$$

where $\alpha = \langle P_c, V_c \rangle(n)$ and the probability is over the coins of both verifiers.

- **Resiliency.** If D rejects at most 2^{m^γ} of its m -bit inputs, then with probability $1 - o(1)$ over the coins r_c of V_c the following holds. For any prover $\tilde{P}_c$, the commitment $\alpha = \langle \tilde{P}_c, V_c \rangle(n)$ generated when V_c uses coins r_c specifies a unique $\tilde{f} \in \{0, 1\}^n$ such that for every $i \in [n]$ and every prover $\tilde{P}_e$,

$$\Pr[\langle \tilde{P}_e, V_e^D \rangle(i, \alpha) \in \{\tilde{f}_i, \perp\}] \geq 1 - 2^{-\Omega(m)},$$

where the probability is over the coins of V_e .

We now present the construction of the targeted generator, combining the generator in Theorem 7.6 and the PCP from Theorem 4.1.

Theorem 7.7 (a targeted HSG based on hardness of sublinear non-deterministic time). *There is a universal $c > 1$ such that the following holds. For any $\varepsilon' < \varepsilon \in (0, 1)$ and $\gamma \in (0, 1)$, let $L \in \text{NTIME}[n^\varepsilon]$ and let $m(n) \triangleq \Theta(n^{\varepsilon'})$. There exists a non-deterministic algorithm H (the generator), and an Arthur-Merlin verifier R (the reconstruction) that for every $x \in \{0, 1\}^n$ satisfy the following.*

- **Generator complexity.** The generator $H(x)$ uses n^ε bits of non-determinism, runs in time $\text{poly}(n)$, and outputs either $\perp$ or a list of $n^{O(\varepsilon)}$ strings of length $m(n)$.
- **Reconstruction complexity.** The reconstruction verifier R gets input x and oracle access to a coAM procedure $D : \{0, 1\}^n \times \{0, 1\}^{m(n)} \rightarrow \{0, 1\}$. Suppose that the underlying deterministic verifier $D(x, y, \rho, w)$ runs in time $T_D(n)$, uses $r_D(n)$ random bits, and uses a $w_D(n)$ -bit witness. The verifier R has one public-coin round of interaction, uses

$$\tilde{O}(n^{2c \cdot \varepsilon \gamma} m(n) r_D(n))$$

³⁵We also assume that V_e receives a representation of $\mathbb{F} = \mathbb{F}_q$ from the prover, otherwise it outputs $\perp$.

bits of public randomness, receives

$$\tilde{O}(n^{2^{c \cdot \varepsilon \gamma}} m(n)(1 + w_D(n)))$$

bits from the prover, and runs in time $\text{poly}(n)$. Its space usage is as follows: (1) It has persistent read-only access to its public random tape; (2) it stores the prover's message; (3) it evaluates $D(x, y, \rho, w)$ on random strings ρ (written on the public random tape) and witnesses w (written on the working tape), reusing the space required for one deterministic evaluation of D ; ³⁶ and (4) it uses $\log^c(n)$ additional bits of space.

Moreover, these deterministic evaluations are the only way in which R accesses its oracle D .

- **Correctness.** Let D be such that

$$\Pr_{y \in \{0,1\}^{m(n)}}[D(x, y) = 0] \leq 2^{m^\gamma - m}.$$

If $x \in L$, then at least one of the following holds:

1. There is a witness such that $H(x)$ does not output $\perp$, and on any such witness it outputs a hitting-set for $D(x, \cdot)$ (i.e., $D(x, y) = 1$ for at least one of the outputs y of $H(x)$).
2. There is an honest prover P such that $\Pr[\langle P, R^D \rangle(x) = L(x)] \geq 1 - 2^{-\Omega(m(n))}$.

If $x \notin L$ then for any $\tilde{P}$ it holds that $\Pr[\langle \tilde{P}, R^D \rangle(x) = 0] \geq 1 - 2^{-\text{polylog}(n)}$.

Proof. Let V be the PCP verifier from Theorem 4.1 for L , instantiated with soundness $s = \mu/2$, and let $N = n^{c \cdot \varepsilon}$ be the proof length for V on n -bit inputs. For $d \triangleq \lceil c \cdot \varepsilon / \varepsilon' \rceil$ and $\gamma > 0$ as in the statement, instantiate $MV = MV_{m, \gamma}$ from Theorem 7.6, and note that $m(N) = \Theta(n^\varepsilon)$.

Generator. The generator $H(x)$ guesses a witness w for $L(x)$, verifies that w is a convincing witness, computes a convincing PCP proof π from w , and outputs $MV(\pi)$. Note that the input length to MV is the PCP proof length $N = n^{c \cdot \varepsilon}$, and thus the number of m -bit strings it outputs is $N^{O(1)} = n^{O(\varepsilon)}$.

Reconstruction. We start by describing a 2-round protocol: The reconstruction R^D on input x first runs the commitment verifier V_c from Theorem 7.6, interacting with the prover on input N to obtain a commitment α , constituting the first round of iteration. It then simulates the PCP verifier V on input x , while answering each of V 's queries $i \in [N]$ to a proof π by running the evaluation verifier $V_e^{D(x, \cdot)}(i, \alpha)$, interacting with the prover to obtain a bit that is supposedly π_i (if the result of this evaluation protocol is $\perp$, the reconstruction rejects). Since V 's queries are all non-adaptive, all these queries are made in parallel, constituting the second round of iteration. The reconstruction accepts if and only if V accepts.

In particular, R^D defines a promise problem $\Pi = (\Pi_Y, \Pi_N)$, where Π_Y consists of the inputs $x \in L$ for which the 2-round protocol has completeness $1 - 2^{-\Omega(m(n))}$, and $\Pi_N = \{x \notin L\}$. We then apply the Babai–Moran round reduction from Lemma 3.10 and convert the protocol into a 1-round protocol over the same promise.

Complexity. The time complexity of H follows from computing a convincing proof π from w and applying Theorem 7.6. The time complexity of R before round reduction is

$$\underbrace{\tilde{O}(N^\gamma)}_{V_c} + \underbrace{\text{poly}(n)}_V + \underbrace{\text{polylog}(n)}_{\text{number of queries by } V} \cdot \underbrace{\text{poly}(m, N^\gamma, r_D, w_D)}_{V_e} < \text{poly}(n).$$

Conducting the Babai–Moran round reduction introduces an additional $\text{polylog}(n) \cdot \tilde{O}(N^\gamma)$ multiplicative factor and remains polynomial time.

As for the space complexity of R , note that:

³⁶The reconstruction evaluates D several times in this manner throughout its execution, but space can be re-used for these simulations.

- In the 2-round protocol, the messages sent by P_c and P_e have lengths

$$\tilde{O}(N^\gamma), \quad \text{polylog}(n) \cdot \tilde{O}(N^\gamma \cdot m(1 + w_D)),$$

respectively. After round reduction, the prover message has length $\tilde{O}(N^{2\gamma}m(1 + w_D))$.

- In the 2-round protocol, the messages sent by V_c and V_e have lengths

$$\tilde{O}(N^\gamma), \quad \text{polylog}(n) \cdot \tilde{O}(N^\gamma \cdot mr_D),$$

respectively. After round reduction, the verifier message has length $\tilde{O}(N^{2\gamma}mr_D)$.

- R only accesses its oracle D when running V_e , where it evaluates the deterministic predicate $D(x, y, \rho, w)$ for randomness ρ and witness w . While the interactive queries by V_e are made in parallel, the space used for accessing its oracle D can be reused.
- The space complexity of simulating V is $\text{polylog}(n)$ (by Theorem 4.1).
- Plugging in $N = n^{c\varepsilon}$ gives the claimed bounds. The public randomness is stored on a persistent read-only tape, and R uses at most $\log^c(n)$ additional space.

Completeness. Let $x \in L$, in which case there is a witness such that $H(x)$ does not output $\perp$ (i.e., a convincing witness for $L(x)$). Assume that on some witness w for $H(x)$, it outputs a set that is not a hitting-set for $D(x, \cdot)$. Let $\pi = \pi(w)$ be the corresponding PCP witness. Then $D(x, y) = 0$ for every $y \in \text{MV}_{m, \gamma}(\pi)$. By the conformity condition in Theorem 7.6, there are provers P_c, P_e such that for every $i \in [N]$,

$$\Pr[\langle P_e, V_e^{D(x, \cdot)} \rangle(i, \alpha) = \pi_i] \geq 1 - 2^{-\Omega(m(n))},$$

where $\alpha = \langle P_c, V_c \rangle(N)$. A union bound over the queries of V shows that the 2-round protocol has completeness $1 - 2^{-\Omega(m(n))}$. Round reduction introduce a multiplicative factor of $2 \cdot \text{polylog}(n) \cdot \tilde{O}(N^\gamma \cdot m(1 + w_D))$ in the completeness error, which is absorbed by $2^{-\Omega(m)}$.

Soundness. Let $x \notin L$. By the density assumption and Theorem 7.6, with probability $1 - o(1)$ over the coins of V_c , there is $\tilde{\pi}$ such that the evaluation protocol answers every query $i \in [N]$ either by $\tilde{\pi}_i$ or by $\perp$, except with probability $2^{-\Omega(m(n))}$. Conditioned on this event, the PCP verifier V accepts x with probability at most s . Thus, the soundness error of the 2-round protocol is at most $s + o(1) + 2^{-\Omega(m(n))} < 1/(2\sqrt{2})$. Performing the Babai–Moran round reduction from Lemma 3.10 reduces the soundness error to $\mu = 2^{-\text{polylog}(n)}$. $\square$

7.2.3 The result

We will be plugging a concrete hard problem in $\text{NTIME}[n^\varepsilon]$ into the generator above. In particular, we will consider the following minimum conditional time-bounded Kolmogorov complexity problem.

Definition 7.8 (Minimum Conditional Time-Bounded Kolmogorov Complexity Problem). *Fix a universal machine U . For a time bound t , define the conditional time-bounded Kolmogorov complexity of x given z by*

$$K^t(x \mid z) \triangleq \min_{\Pi} \left\{ |\Pi| : U(\Pi(z), 1^{t(|x|)}) = x \right\}.$$

For polynomials t and ζ , the Minimum Conditional Time-Bounded Kolmogorov Complexity Problem is the language

$$\text{McK}^t\text{P}[\zeta] \triangleq \bigcup_{n \geq 1} \left\{ x \circ z \circ k : x \in \{0, 1\}^n, z \in \{0, 1\}^{\zeta(n)}, k \in \{0, 1\}^{\lceil \log n \rceil}, K^t(x \mid z) \leq k \right\}.$$

Lemma 7.9 (conditional time-bounded Kolmogorov complexity in arbitrarily small non-deterministic time). *Fix a constant $\varepsilon > 0$ and a polynomial time bound t satisfying $t(n) \geq (1 + \varepsilon)n$. For every constant $\eta > 0$, there is an integer $q = O_t(1/\eta)$ such that, for $\zeta(n) = n^q$, the language $\text{McK}^t\text{P}[\zeta]$ belongs to $\text{NTIME}[N^\eta]$, where $N = n + n^q + \lceil \log n \rceil$ denotes the total input length.*

Proof. Consider the following non-deterministic algorithm: First, the algorithm recovers n from $N = n + n^q + \lceil \log n \rceil$ in $\text{polylog}(N)$ time, rejecting if no such n exists. On input $x \circ z \circ k$, it non-deterministically guesses a program Π of length at most k , simulates its $t(n)$ -step computation on z , and accepts exactly when it outputs x . Correctness follows from Definition 7.8. Since $k < 2n$ and the algorithm has random access to z , this takes $n^d + \text{polylog}(N)$ time for a fixed constant d depending only on t . Choosing $q > d/\eta$ makes the running time $O(N^\eta)$ as desired. $\square$

This problem is particularly interesting because its average-case hardness can be derived from the existence of one-way functions.

Theorem 7.10 (Liu–Pass reduction [LP20, LP22]). *For any candidate one-way function f there is a constant $c = c_f \in \mathbb{N}$ such that the following holds. Assume that for a polynomial $t(n) \geq 1.01 \cdot n$ and $\eta(n) = n^q$ there is a probabilistic algorithm H running in time $T(N)$ such that for all sufficiently large n ,*

$$\Pr_{\substack{x \in \{0,1\}^n, \\ z \in \{0,1\}^{\zeta(n)}, \\ k \in \{0,1\}^{\lceil \log n \rceil}, H}} [H(x \circ z \circ k) = \text{McK}^t\text{P}[\zeta](x, z, k)] \geq 1 - \frac{1}{n^c}.$$

Then there is a probabilistic I running in time $T'(\lambda) = \text{poly}(T(\text{poly}(\lambda)))$ such that for all sufficiently large λ

$$\Pr_{x \in \{0,1\}^\lambda, I} [I(1^\lambda, f(x)) \in f^{-1}(x)] \geq 1/\text{poly}(\lambda).$$

Theorem 7.11 (average-case targeted HSG from cryptography). *Assume that there are sub-exponentially secure one-way functions. Then there are constants $\varepsilon \in (0, 1)$ and $c > 0$ and a non-deterministic polynomial-time algorithm H that, on input $x \in \{0, 1\}^n$, uses at most n^ε non-deterministic bits and outputs, on every accepting path, $L(n) = n^{O(\varepsilon)}$ strings of length $m(n) = \Theta(n^{\varepsilon^2})$ such that the following holds.*

For every coAM procedure $D: \{0, 1\}^n \times \{0, 1\}^{m(n)} \rightarrow \{0, 1\}$ whose deterministic verifier runs in time $T_D(n) = 2^{n^{\varepsilon^2}}$ and uses $r_D(n) = w_D(n) = n^{\varepsilon^3}$ random and witness bits, the following holds. For infinitely many n , with probability at least n^{-c} over $x \in \{0, 1\}^n$, if

$$\Pr_{y \in \{0,1\}^{m(n)}} [D(x, y) = 1] \geq 1 - 2^{m(n)^\varepsilon - m(n)},$$

then $H(x)$ has an accepting path, and on every accepting path its output contains a string y with $D(x, y) = 1$.

Proof. Fix a sub-exponentially secure one-way function and a polynomial $t \geq 1.01 \cdot n$ as in Theorem 7.10. By Lemma 7.9, for every sufficiently small $\varepsilon > 0$ we can choose $q = O(1/\varepsilon)$ such that, for $\zeta(u) = u^q$, the language $L^{\text{hard}} = \text{McK}^t\text{P}[\zeta]$ belongs to $\text{NTIME}[n^\varepsilon]$, where n denotes total input length. Theorem 7.10 then gives constants $\delta = \Omega(\varepsilon)$ and $a > 0$ such that every probabilistic 2^{n^δ} -time algorithm for L^{hard} errs with probability at least $n^{-c''}$ over a choice of n -bit inputs and its internal randomness, for infinitely many n . We define $c = 2c''$ (any constant larger than c'' would do).

Apply Theorem 7.7 to L^{hard} with $\varepsilon' = \varepsilon^2$ and $\gamma = \varepsilon$. This gives a generator H with the claimed complexity and output parameters. For a procedure D as in the statement, the reconstruction's prover message has length

$$\tilde{O}\left(n^{2c'\varepsilon^2} m(n)(1 + w_D(n))\right) = \tilde{O}\left(n^{(2c'+1)\varepsilon^2 + \varepsilon^3}\right),$$

where c' is the constant in Theorem 7.7.

Fix D , and assume towards a contradiction that for all sufficiently large n , on more than $1 - n^{-c}$ fraction of inputs, H fails to hit D . Theorem 7.7 therefore guarantees that the reconstruction protocol R correctly decides L^{hard} on these $1 - n^{-c}$ fraction of inputs.

We turn the reconstruction protocol R into a probabilistic algorithm by sampling its public randomness, enumerating every prover message. Recall that R accesses D only on inputs (x, y) where $D(x, y)$ rejects, in which case it samples public randomness ρ and the prover sends a string w witnessing $D(x, y, \rho, w) = 0$.

The algorithm accepts if any enumeration of the prover message makes the verifier accept. On each of the $1 - n^{-c}$ fraction of inputs above, completeness and soundness of R imply that one run errs with probability at most $1/3$. Repeating this algorithm independently $O(\log n)$ times and taking the majority answer reduces the error to at most n^{-c} .

Repetition increases the running time by only an $O(\log n)$ factor and the total running time remains $2^{n^{O(\varepsilon^2)}} \leq 2^{n^\delta}$ for sufficiently small ε .

The error is at most n^{-c} on each of the inputs above; the remaining n^{-c} fractions of inputs (on which H hits D) contribute at most another n^{-c} error. Thus the error over the input distribution and the internal randomness of the algorithm is at most $2n^{-c} < n^{-c''}$ for all sufficiently large input lengths, contradicting Theorem 7.10. $\square$

Acknowledgements

We thank Swastik Kopparty for helpful discussions in the early stage of this project. We thank Hanlin Ren for telling us about the average-case algorithm of [HIR25] for a special case of low-space Avoid, which we originally suspected may be a candidate hard instance. We also thank Valentine Kabanets for pointing out Corollary 5.15. We are grateful to Gal Arnon, Oliver Korten, and Ryan Williams for useful comments and feedback about an early draft of this paper.

Part of this work was done while the authors were participating in the workshop “Frontiers in complexity lower bounds” at the Isaac Newton Institute for Mathematical Sciences, Cambridge. Part of this work was also done while the second author was participating in the program “Pseudorandomness & High-Dimensional Expansion” at the Simons Institute for the Theory of Computing.

AI Usage

This paragraph is included as part of the initial collective adaptation of TCS to AI usage, and in particular following the recommendations of the report “AI and TCS: The Next Six Months” from September 2026. GPT 5.6 Sol and GPT 6 Astra were used in the preparation of the manuscript, although almost all technical ideas were developed without AI usage. Usage of AI includes looking up relevant literature, and in particular finding the Buhrman-Torenvliet paper [BT00]; drafting preliminary versions of proof texts in response to prompts that include the main proof ideas; and copy-editing.

References

- [ABCY22] Gal Arnon, Amey Bhangale, Alessandro Chiesa, and Eylon Yogev. A toolbox for barriers on interactive oracle proofs. In *Theory of cryptography. Part I*, volume 13747 of *Lecture Notes in Comput. Sci.*, pages 447–466. 2022. 51
- [AF09] Luís Antunes and Lance Fortnow. Worst-case running times for average-case algorithms. In *Proc. 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 298–303. 2009. 52
- [ANOS24] Bar Alon, Moni Naor, Eran Omri, and Uri Stemmer. MPC for tech giants (GMPC): enabling Gulliver and the Lilliputians to cooperate amicably. In *Advances in cryptology—CRYPTO 2024. Part VIII*, volume 14927 of *Lecture Notes in Comput. Sci.*, pages 74–108. [2024] ©2024. 17

- [BBH⁺19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of Kilian-based SNARGs. In *Theory of cryptography. Part II*, volume 11892, pages 522–551. 2019. [17](#)
- [BCT25] Marshall Ball, Lijie Chen, and Roei Tell. Towards free lunch derandomization from necessary assumptions (and OWFs). In *Proc. 40th Annual IEEE Conference on Computational Complexity (CCC)*, pages Art. No. 31, 20. 2025. [40](#)
- [BGH⁺05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proc. 20th Annual IEEE Conference on Computational Complexity (CCC)*, pages 120–134, 2005. [10](#), [17](#)
- [BM88] László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988. [9](#), [15](#), [23](#)
- [BR22] Liron Bronfman and Ron D. Rothblum. PCPs and instance compression from a cryptographic lens. In *Proc. 13th Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 215, pages Art. No. 30, 19. 2022. [17](#)
- [BT00] Harry Buhrman and Leen Torenvliet. Randomness is hard. *SIAM Journal on Computing*, 30(5):1485–1501, 2000. [7](#), [46](#)
- [CHLR25] Lijie Chen, Shuichi Hirahara, Zeyong Li, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. *J. ACM*, November 2025. [2](#)
- [CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *Advances in cryptography—CRYPTO 2011*, volume 6841 of *Lecture Notes in Comput. Sci.*, pages 151–168. 2011. [17](#)
- [CL24] Yilei Chen and Jiayu Li. Hardness of range avoidance and remote point for restricted circuits via cryptography. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 620–629, 2024. [2](#), [3](#), [4](#)
- [CLL25] Lijie Chen, Jiayu Li, and Jingxun Liang. Maximum circuit lower bounds for exponential-time Arthur Merlin. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1348–1358, 2025. [2](#)
- [CM24] James Cook and Ian Mertz. Tree evaluation is in space $O(\log n \cdot \log \log n)$. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1268–1278, 2024. [1](#)
- [CRT25] Lijie Chen, Ron D. Rothblum, and Roei Tell. Fiat-Shamir in the plain model from derandomization (or: do efficient algorithms believe that $\text{NP} = \text{PSPACE}$?). In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 977–985, [2025] ©2025. [8](#), [52](#)
- [CSW06] Mahdi Cheraghchi, Amin Shokrollahi, and Avi Wigderson. Computational hardness and explicit constructions of error correcting codes. In *Proc. 44th Allerton Conference on Communication, Control and Computing, Allerton House, IL, USA*, 2006. [51](#)
- [CT21a] Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2021. [7](#), [39](#), [40](#)
- [CT21b] Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: Eliminating randomness at almost no cost. In *Proc. 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 283–291, 2021. [40](#)

- [CT22] Lijie Chen and Roei Tell. When Arthur has neither random coins nor time to spare: Superfast derandomization of proof systems. *Electronic Colloquium on Computational Complexity: ECCC*, 2022. [8](#), [17](#)
- [CT23a] Lijie Chen and Roei Tell. Guest column: New ways of studying the $\mathbf{BPP} = \mathbf{P}$ conjecture. *ACM SIGACT News*, 54(2):44–69, 2023. [39](#), [52](#)
- [CT23b] Lijie Chen and Roei Tell. When Arthur has neither random coins nor time to spare: Superfast derandomization of proof systems. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 60–69, 2023. [40](#)
- [DMOZ22] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *Journal of the ACM*, 69(6):1–55, 2022. [40](#)
- [DMOZ26] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Pseudorandomness beating the hybrid argument for insensitive algorithms. *Electronic Colloquium on Computational Complexity: ECCC*, 2026. [40](#)
- [DPT24] Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for $\mathbf{BPL} = \mathbf{L}$ that uses properties of $\mathbf{BPL}$. In *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*, 2024. [40](#)
- [DT23] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. [40](#)
- [GLS26] Surendra Ghentiyala, Zeyong Li, and Noah Stephens-Davidowitz. Range avoidance, Arthur-Merlin, and TFNP. In *Proc. 58th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1925–1936, 2026. [2](#)
- [Gol11a] Oded Goldreich. In a world of $\mathbf{P} = \mathbf{BPP}$. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay Randomness and Computation*, pages 191–232, 2011. [5](#), [39](#)
- [Gol11b] Oded Goldreich. Two comments on targeted canonical derandomizers. *Electronic Colloquium on Computational Complexity: ECCC*, 2011. [39](#)
- [GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, 1986. [13](#), [15](#)
- [GSTS03] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. Uniform hardness versus randomness tradeoffs for Arthur-Merlin games. *Computational Complexity*, 12(3-4):85–130, 2003. [41](#)
- [GZ11] Oded Goldreich and David Zuckerman. Another proof that $\mathbf{BPP} \subseteq \mathbf{PH}$ (and more). volume 6650 of *Lecture Notes in Computer Science*, pages 40–53. 2011. [27](#)
- [Har04] Prahladh Harsha. *Robust PCPs of Proximity and Shorter PCPs*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, September 2004. Ph.D. thesis, Department of Electrical Engineering and Computer Science. Advisor: Madhu Sudan. [10](#), [17](#)
- [HIR25] Yizhi Huang, Rahul Ilango, and Hanlin Ren. NP-hardness of approximating meta-complexity: a cryptographic approach. *SIAM Journal on Computing*, 54(4):819–886, 2025. [46](#)
- [HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Proc. 23rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 672–683. 2006. [31](#)

- [IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002. [5](#)
- [Ila25] Rahul Ilango. The oracle derandomization hypothesis is false (and more) assuming no natural proofs. *Electronic Colloquium on Computational Complexity: ECCC*, 32:190, 2025. [2](#), [3](#), [4](#), [5](#), [12](#), [13](#), [33](#), [34](#)
- [ILW23] Rahul Ilango, Jiayu Li, and R. Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proc. 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1076–1089, [2023] ©2023. [2](#), [4](#)
- [KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos H. Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, volume 185 of *LIPICs*, pages 44:1–44:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. [2](#)
- [KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs and PPAD hardness from the decisional Diffie-Hellman assumption. In *Proc. Advances in cryptology (EUROCRYPT)*, pages 470–498. [2023] ©2023. [32](#)
- [Kor21] Oliver Korten. The hardest explicit construction. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 433–444. IEEE, 2021. [2](#)
- [Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *Proc. 37th Annual IEEE Conference on Computational Complexity (CCC)*, 2022. [7](#)
- [Kor25] Oliver Korten. Range avoidance and the complexity of explicit constructions. *Bulletin of the European Association for Theoretical Computer Science. EATCS*, (145):94–134, 2025. [2](#)
- [Koz78] Dexter Kozen. Indexing of subrecursive classes. In *Proc. 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 287–295, 1978. [1](#), [7](#)
- [KS25] Oliver Korten and Rahul Santhanam. How to construct random strings. In *Proc. 40th Annual IEEE Conference on Computational Complexity (CCC)*, pages Art. No. 35, 32. 2025. [5](#), [7](#)
- [KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002. [3](#), [10](#), [25](#), [26](#)
- [LOZ22] Zhenjian Lu, Igor C. Oliveira, and Marius Zimand. Optimal coding theorems in time-bounded Kolmogorov complexity. In *Proc. 49th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 229, pages Art. No. 92, 14. 2022. [52](#)
- [LP20] Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. In *Proc. 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1243–1254. [2020] ©2020. [11](#), [13](#), [38](#), [41](#), [45](#)
- [LP22] Yanyi Liu and Rafael Pass. On one-way functions from np-complete problems. In *Proc. 37th Annual IEEE Conference on Computational Complexity (CCC)*, pages 36:1–36:24, 2022. [11](#), [13](#), [38](#), [41](#), [45](#)
- [Mil01] Peter Bro Miltersen. Derandomizing complexity classes. In *Handbook of randomized computing, Vol. I, II*, volume 9 of *Comb. Optim.*, pages 843–941. Kluwer Acad. Publ., Dordrecht, 2001. [51](#)
- [MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007. [11](#)

- [MV05] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005. 13, 41
- [Nep70] V. Nepomnjascii. Rudimentary predicates and turing calculations. *Soviet Math. Doklady*, 11:1462–1465, 1970. 10, 15, 26
- [NIR03] Alan Nash, Russell Impagliazzo, and Jeff Remmel. Universal languages and the power of diagonalization. In *Proc. 18th Annual IEEE Conference on Computational Complexity (CCC)*, page 337, 2003. 7
- [NIR06] Alan Nash, Russell Impagliazzo, and Jeff Remmel. Infinitely-often universal languages and diagonalization. *Electronic Colloquium on Computational Complexity: ECCC*, 13:51, 2006. 7
- [Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. 27
- [Nis93] Noam Nisan. On read-once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993. 3
- [NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. 13
- [Ope26] OpenAI. Simulating one-tape time in two-fifths-power space. <https://github.com/openai/math/blob/main/preprints/Simulating-One-Tape-Time-in-Two-Fifths-Power-Space-September-25-2026/article.pdf>, 2026. Accessed: 2026-10-07. 1
- [PR23] Rafael Pass and Oren Renard. Characterizing the power of (persistent) randomness in log-space. *Electronic Colloquium on Computational Complexity: ECCC*, 2023. 3
- [RW26] Hanlin Ren and Ryan Williams. Near-maximum circuit lower bounds for exponential time with merlin-arthur queries. In *Proc. 67th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2026. 2
- [RWZ26] Hanlin Ren, Yichuan Wang, and Yan Zhong. Hardness of Range Avoidance and Proof Complexity Generators from Demi-Bits. In Shubhangi Saraf, editor, *17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*, volume 362 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 111:1–111:25, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 2, 4
- [San23] Rahul Santhanam. An algorithmic approach to uniform lower bounds. In *Proc. 38th Annual IEEE Conference on Computational Complexity (CCC)*, pages Art. No. 35, 26. 2023. 7
- [Sha24] Yakov Shalunov. Leakage-resilient hardness equivalence to logspace derandomization. In *Proc. 49th International Symposium on Mathematical Foundations of Computer Science*, pages Art. No. 83, 16. 2024. 3
- [Sip88] Michael Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, 1988. 52
- [SV22] Ronen Shaltiel and Emanuele Viola. On hardness assumptions needed for “extreme high-end” PRGs and fast derandomization. In *Proc. 13 Conference on Innovations in Theoretical Computer Science (ITCS)*, 2022. 40
- [Vad12] Salil P. Vadhan. *Pseudorandomness*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2012. 26

- [vMS23a] Dieter van Melkebeek and Nicollas Sdroievski. Instance-wise hardness versus randomness tradeoffs for arthur-merlin protocols. In *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*, 2023. [5](#), [13](#), [39](#)
- [vMS23b] Dieter van Melkebeek and Nicollas Sdroievski. Leakage resilience, targeted pseudorandom generators, and mild derandomization of Arthur-Merlin protocols. In *Proc. 43rd Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. 2023. [5](#), [39](#)
- [VZ15] N. V. Vinodchandran and Marius Zimand. On optimal language compression for sets in PSPACE/poly. *Theoretical Computer Science*, 56(3):581–590, 2015. [51](#)
- [Wil25] R. Ryan Williams. Simulating time with square-root space. In *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*, pages 13–23, 2025. [1](#), [2](#), [6](#), [16](#), [30](#), [31](#), [52](#)
- [Zim11] Marius Zimand. On the optimal compression of sets in PSPACE. In *Fundamentals of computation theory*, volume 6914 of *Lecture Notes in Comput. Sci.*, pages 65–77. Springer, Heidelberg, 2011. [51](#)

A Prior Assumptions that Do Not Coincide with Low-Space Simulation of FP

For completeness, we mention several prior works in complexity theory and cryptography that relied on hardness assumptions that contradict low-space simulation of FP (i.e., as in Theorem 1.2).

As mentioned in Section 1, there are actually remarkably *few* works making such assumptions. The works mentioned below (with the exception of the first one) did not study the assumption that time cannot be simulated in polynomially smaller space, but rather relied on it to instantiate hardness vs randomness tradeoffs. In all of these works the conclusion may still be true (even if the assumption is false), and in most of these works, the assumption can be relaxed and/or was assumed mostly for simplicity. Let us elaborate.

Studying the assumption itself. Miltersen [Mil01] showed that the assumption that E is hard for $\text{SPACE}[2^{\varepsilon \cdot n}]$, for some $\varepsilon > 0$, actually implies that E is hard for *non-uniform* circuits of exponential size with PSPACE gates. That is, hardness for uniform bounded-space implies hardness for non-uniform bounded-space.

Compressing instances of problems in PSPACE. Zimand [Zim11] and Vinodchandran and Zimand [VZ15] assumed that $\text{E} \not\subseteq \text{i.o.-SPACE}[2^{\varepsilon \cdot n}]$, for some $\varepsilon > 0$, to deduce optimal compression of instances of PSPACE-languages (the conclusion improves on a line of prior works studying such compression).

Derandomizing interactive oracle proofs. Arnon, Bhangale, Chiesa, and Yogeve [ABCY22] assumed a pseudorandom generator against non-uniform circuits with PSPACE gates in order to derandomize interactive oracle proofs (IOPs). As pointed out by Gal Arnon, their work also obtains similar derandomization without the hardness assumption, but using non-uniform advice (see [ABCY22, Item (2) of Lemma 3]).

Using hardness-vs-randomness to construct codes. Cheraghchi, Shokrollahi and Wigderson [CSW06] used the same assumption as [Zim11, VZ15] to construct explicit codes meeting the GV bound. However, their text is expository in nature (as they mentioned, their main point is to “observ[e] that [the underlying techniques] are relevant to the construction of optimal codes”).

Works in which the assumption was stated but can be replaced or discarded. In several works the assumption that time is hard for smaller space was stated, but it is already known that the conclusion in the work can be deduced from weaker assumptions. In particular:

1. Antunes and Fortnow [AF09] used hardness for non-uniform circuits with PSPACE gates to prove a two-way connection between the worst-case running time of any algorithm A and its running time on inputs from samplable distributions.³⁷ Later on the same conclusion was proved *unconditionally* by Lu, Oliveira, and Zimand [LOZ22].
2. Chen, Rothblum, and Tell [CRT25] presented an assumption that FP is hard for sub-linear space on average, in order to derandomize interactive proof systems into what they called “computationally sound NP”. However, they noted that this assumption was presented for simplicity, and that their proof also goes through when the hard function is in NP (and even in computationally sound NP; see [CRT25, Theorem 1.3]).
3. A classical work of Sipser [Sip88] deduced $\text{RP} = \text{P}$ from the existence of expanders (which we now know how to unconditionally construct) and from time T being hard for smaller space $T^{1-\varepsilon}$ for all $\varepsilon > 0$ (where the latter is false, by [Wil25]). Of course, given the rich literature on hardness vs randomness, we now know how to deduce $\text{RP} = \text{P}$ from various other assumptions (for a recent survey, see [CT23a]).

³⁷Already in the original paper, they noted that their proof goes through even when hardness is for circuits with Σ_2 -gates rather than PSPACE gates, and mentioned that a further weakening was obtained by van Melkebeek.