When Connectivity Is Hard, Random Walks Are Easy With Non-Determinism

Dean Doron
Ben-Gurion University
deand@bgu.ac.il

Edward Pyne
MIT
epyne@mit.edu

Roei Tell
University of Toronto
roei@cs.toronto.edu

R. Ryan Williams
MIT
rrw@mit.edu

Abstract

Two fundamental problems on directed graphs are to decide s-t connectivity, and to estimate the behavior of random walks. Currently, there is no known algorithm for s-t connectivity running in polynomial time and $n^{o(1)}$ space, and no known algorithm for estimating the n-step random walk matrix running in non-deterministic logspace.

We show that for every directed graph, at least one of these problems is solvable in time and space that significantly improve on the respective state-of-the-art. In particular, there is a pair of algorithms A_1 and A_2 such that for every graph G, either:

1. $A_1(G)$ outputs the transitive closure of G in polynomial time and polylogarithmic space.
2. $A_2(G)$ outputs an approximation of the n-step random walk matrix of G in non-deterministic logspace.

As one application, we show surprisingly tight win-win results for space-bounded complexity. For example, for certain parameter regimes, either Savitch's theorem can be non-trivially sped up, or randomized space can be almost completely derandomized.

We also apply our techniques to significantly weaken the assumptions required to derandomize space-bounded computation, and to make non-deterministic space-bounded computation unambiguous. Specifically, we deduce such conclusions from lower bounds against uniform circuits of polynomial size, which is an exponential improvement on the required hardness in previous works (Doron–Pyne–Tell STOC 2024, Li–Pyne–Tell FOCS 2024). We further show similar results for minimal-memory derandomization (Doron–Tell CCC 2024).

To prove these results, we substantially improve the array of technical tools introduced in recent years for studying hardness-vs.-randomness for bounded-space computation. In particular, we develop derandomized distinguish-to-predict transformations for new types of distinguishers (corresponding to compositions of PRGs with weak distinguishers), we construct a derandomized logspace reconstruction procedure for the Shaltiel–Umans generator (JACM 2005) that can compress hard truth-tables to polylogarithmic size, and we design a version of the Chen–Tell generator (FOCS 2021) that is particularly suitable for the space-bounded setting.

Contents

1 Introduction

How much time and space is necessary to simulate randomized small-space algorithms? Or, alternatively, to simulate non-deterministic small-space algorithms? These are two of the most well-studied questions in space complexity, with particularly clean complete problems:

(1) For $\mathbf{BPL} = \mathbf{BPSPACE}[O(\log n)]$, the corresponding problem is to estimate n-step random walk probabilities on an n-vertex graph, with small additive error.

(2) For $\mathbf{NL} = \mathbf{NSPACE}[O(\log n)]$, the problem is to decide s-t connectivity on an n-vertex graph.

At the moment, we do not know how to solve either problem in polynomial time and only logarithmic space. For problem (1), it is widely conjectured that such an algorithm exists (i.e., that $\mathbf{BPL} = \mathbf{L}$, which follows from conjectured lower bounds [KvM02, DT23, DPT24]). However, the best known algorithms work either in super-polynomial time $2^{\log(n)^{3/2-o(1)}}$ and in space $\log(n)^{3/2-o(1)}$ [SZ99, Hoz21], or in polynomial time and in larger space $O(\log^2 n)$ [Nis92, CCvM06]. Moreover, it is not even known how to improve these algorithms using non-deterministic computation (e.g., we do not know whether $\mathbf{BPL} \subseteq \mathbf{NSPACE}[\log^{1.49} n]$).

For problem (2), there is seemingly no consensus on a widely believed conjecture, and the most important reference point is the classical algorithm of Savitch [Sav70], which works in super-polynomial time $2^{\Theta(\log^2 n)}$ and in space $O(\log^2 n)$. A major open problem asks whether Savitch's algorithm can be improved. For example, already 30 years ago, Wigderson [Wig92] asked whether an algorithm can achieve polynomial time and space $n^{1-\varepsilon}$ for some $\varepsilon > 0$: This problem is still wide open, and the best known polynomial-time algorithm uses nearly-linear space $n/2^{\Theta(\sqrt{\log n})}$ [BBRS98]. In fact, in a natural restricted model encompassing all known deterministic, randomized, and non-deterministic algorithms for directed (and undirected) connectivity,[1] there is a lower bound ruling out algorithms running in time $2^{\log^{1.99} n}$ and space $n^{0.99}$ [EPA99].

1.1 Our Results, Part 1: A Pair of Algorithms

We prove that for every graph, *at least one* of these problems can be solved significantly more efficiently than previously known algorithms:

Theorem 1. *There are algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for every graph G on n vertices, one of the following holds:*

- *$\mathcal{A}_1(G)$ solves s-t connectivity in G in polynomial time and polylogarithmic space.*
- *$\mathcal{A}_2(G)$ estimates length-n random walk probabilities in G in non-deterministic logspace.*[2]

Moreover, both algorithms report if they fail to compute the desired answer, and do not exceed their resource bounds in any case.

[1] Specifically, this is the Node-Named Jumping Automata on Graphs (NNJAG) model [CR80, Poo93, LZPC05]. This model captures all known space-bounded directed and undirected connectivity algorithms, including Savitch, BFS, DFS, Immerman-Szelepcsényi [Imm88, Sze88], Nisan et al. [NSW92], Barnes et al. [BBRS98], Armoni et al. [ATWZ00], and Reingold [Rei08].

[2] The estimation is up to an additive $1/\operatorname{poly}(n)$ error. The algorithm runs in logspace, makes non-deterministic guesses, and either declares *fail* if the guess sequence is bad, or a single canonical matrix only depends on the graph G, or a special symbol $\perp$ indicating that $\mathcal{A}_2$ does not succeed on this input (in which case $\mathcal{A}_1$ succeeds on the input).

We stress that the constructions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are explicit (i.e., these are specific algorithms, and their descriptions will be given in Section 2.1), and that provably, for *every graph*, at least one of the algorithms works (and both of them never return an incorrect answer). Moreover, these algorithms run in time and space that *significantly* improve on the respective state-of-the-art: The algorithm $\mathcal{A}_1$ runs in polynomial time and uses only polylogarithmic space (compared to $n/2^{\sqrt{\log n}}$ space [BBRS98]); and $\mathcal{A}_2$ uses only logarithmic space, alas it also uses non-determinism (i.e., it is akin to $\mathsf{BPL} \subseteq \mathsf{NL}$; in comparison, the algorithm of [SZ99, Hoz21] uses $\log(n)^{3/2-o(1)}$ space).

Application: Tight win-win results in space-bounded complexity. Since the pair of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ from Theorem 1 solve the complete problems for for NL and for BPL, respectively, we can use them to tightly connect the challenges of simulating NL and BPL. As one application, we leverage the pair of algorithms to show that either Savitch's theorem can be improved, or randomized space can be deterministically simulated *near-optimally*.

Theorem 2. *For every constant $\varepsilon > 0$, at least one of the following holds:*

- $\mathsf{NSPACE}[n] \subseteq i.o.\mathsf{TISP}\left[2^{O(n^{2-\varepsilon})}, n^{O(1)}\right]$.
- $\mathsf{BPSPACE}[n] \subseteq \mathsf{SPACE}\left[O(n^{1+\varepsilon})\right]$.

We stress that the second item in Theorem 2 does *not* use non-determinism (i.e., it is a scaled-up version of $\mathsf{BPL} \subseteq \mathsf{SPACE}[(\log n)^{1+\varepsilon})]$), in contrast to $\mathcal{A}_2$ from Theorem 1. Indeed, Theorem 2 is not an immediate corollary of Theorem 1, but it does use the techniques underlying the proof of the latter (i.e., a more general construction of a pair of algorithms).

Theorem 2 is particularly meaningful in two parameter regimes, corresponding to choices of $\varepsilon > 0$. Specifically, the following two instantiations assert that for each of the two problems we consider (i.e., s-t connectivity and estimating random walks), either we can non-trivially improve on the state-of-the-art for solving the problem, or we can *near-optimally* solve the other problem:

- With an arbitrarily small $\varepsilon > 0$: Either Savitch's algorithm can be non-trivially sped-up (i.e., replacing 2^{n^2} with $2^{n^{2-\varepsilon}}$), or we can near-optimally derandomize $\mathsf{BPSPACE}[n]$.
- With $\varepsilon = 0.49$: Either the frontier derandomization of Saks–Zhou can be non-trivially improved (i.e. replacing $\mathsf{SPACE}[n^{1.5}]$ with $\mathsf{SPACE}[n^{1.49}]$), or Savitch's algorithm can be substantially sped up (i.e. replacing 2^{n^2} with $2^{n^{3/2+0.01}}$).

If we are willing to settle for non-deterministic simulation of $\mathsf{BPSPACE}$, we can leverage Theorem 1 to connect *near-optimal* solutions to *both* problems (i.e., rather than connecting a slight improvement to the state-of-the-art for one problem to a near-optimal solution to the other problem). For example, either Savitch's Theorem can be optimally sped up, or we can optimally simulate probabilistic linear space using non-determinism:

Theorem 3 (informal; see Theorem 6.10)**.** *It holds that either* $\mathsf{NSPACE}[n] \subseteq i.o.\mathsf{TISP}\left[2^{O(n)}, n^{O(1)}\right]$, *or* $\mathsf{BPSPACE}[n] \subseteq \mathsf{NSPACE}[O(n)]$.

Interpretation. If one believes that $\mathsf{L} = \mathsf{NL}$ and $\mathsf{BPL} = \mathsf{L}$ (i.e., that we can solve s-t connectivity in logspace and estimate random walk probabilities in logspace), then Theorem 1 and the win-win results can be interpreted as concrete steps towards proving *both* statements. Alternatively, if one believes that (say) Savitch's algorithm cannot be sped up, then our results can be interpreted as showing that such a statement implies optimal derandomization. In any case, our results in this section provide a new algorithmic tool, and connect two fundamental problems.

1.2 Our Results, Part 2: Derandomization from Very Weak Hardness

One perspective on the win-win results above is that they convert hardness into randomness: Specifically, they deduce near-optimal derandomization of small space from hardness of improving Savitch's theorem (i.e., of solving s-t connectivity) by uniform, deterministic algorithms. We stress that typical results in hardness vs. randomness deduce derandomization from stronger assumptions (i.e., from hardness for non-uniform circuits or for probabilistic algorithms).[3]

From this perspective (i.e., if the goal is to deduce derandomization from weak hardness), we want to do better than Theorem 2 and Theorem 3, by deducing optimal derandomization – without non-determinism, and without an $n^{1+\varepsilon}$ space overhead. We are indeed able to do so.

Full derandomization of BPSPACE from very weak hardness. A classical result of Klivans and van Melkebeek [KvM02] (following [NW94]) showed that $\mathbf{BPL} = \mathbf{L}$ follows from sufficiently explicit lower bounds against exponential sized non-uniform circuits (see also [DT23]). Since these circuit lower bounds currently seem out of reach, a natural direction is to deduce derandomization from assumptions that are weak enough so that we hope to unconditionally prove them.

Recently, Doron, Pyne, and Tell [DPT24] showed one such result, in which they deduced derandomization from lower bounds for exponential-sized circuits that can be printed by *uniform, space-bounded machines* (rather than non-uniform circuits). As they point out, proving such a lower bound seems significantly more tractable , since there are already known lower bounds against uniform circuits (see, e.g., Santhanam and Williams [SW13]).

To be more precise, in [DPT24] they proved that $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[O(n)]$ follows from hardness of $\mathbf{SPACE}[n]$ against logspace-uniform oracle circuits of size $2^{\varepsilon n}$; that is, against exponential-sized circuits that can be printed in space $O(n)$. We deduce the same conclusion from lower bounds against uniform *polynomial-sized circuits* (equipped with an oracle that uses space εn).

Theorem 4. *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that $\mathbf{SPACE}[n]$ is hard for $\mathbf{TISP}[2^{cn}, n^c]$-uniform circuits of size n^c with oracle access to $\mathbf{SPACE}[\varepsilon n]$.*[4] *Then, $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[O_\varepsilon(n)]$.*

Theorem 4 represents a near-exponential improvement in the size of the circuits against which we need hardness, at the cost of relaxing the uniformity condition from $\mathbf{SPACE}[O(n)]$-uniformity to $\mathbf{TISP}[2^{O(n)}, n^{O(1)}]$-uniformity. The assumption in Theorem 4 strikes us as very weak, and plausibly provable: It asserts that there are N-bit strings printable in space $O(\log N)$ that cannot be deterministically compressed (in small time and space) to a circuit of size $\mathrm{polylog}(N)$ that can make oracle queries to space $\varepsilon \cdot \log(N)$.

Derandomization with minimal memory footprint. We also consider the question of derandomization with minimal memory overhead, which was introduced by Doron and Tell [DT23] (following [DMOZ22, CT21b]). The classical conjecture $\mathbf{BPL} = \mathbf{L}$ asserts that randomized space-S machines can be simulated in space $S' = C \cdot S$, for some (possibly large) constant $C > 1$. The more ambitious goal from [DT23] is to have S' as close as possible to S; for example, deduce simulation with $S' \approx 2S$ or even $S' \approx S$. Since we do not hope to show this unconditionally at the moment, the goal is to deduce it under the weakest possible assumptions.

[3] Only very recently, several works deduced derandomization of small-space computation from hardness for uniform, deterministic algrithms (see [PRZ23, DPT24, LPT24]). Jumping ahead, we build on these works and significantly develop the technical machinery introduced in them. For details, see Section 1.3.

[4] The input length n to the oracle is the same length as the input to the generating algorithm (so we do not let the machine write longer oracle queries).

We base minimal-memory derandomization on assumptions that are qualitatively weaker than those known to imply *standard* (i.e. not superfast) derandomization in the time-bounded setting. To see this, recall that the work of [DT23] deduced derandomization of randomized space-S with deterministic space $S' = 2S + O(\log n)$ under two assumptions: very efficient cryptographic PRGs, and strong circuit lower bounds. The subsequent work [DPT24] obtained the same conclusion without the cryptographic assumption, while still requiring lower bounds for non-uniform circuits. We go further, deducing the same conclusion from hardness of compression of a multi-output function by *uniform, deterministic* machines that run in polynomial time and sublinear space:

Theorem 5 (see Theorem 6.14)**.** *Assume that for any large enough constant C there exists a function f mapping n bits to n^2 bits that is computable in space $(C+1)\cdot\log(n)$, but for any deterministic algorithm R that runs in space $n^{0.01}$ and time $n^{O(C)}$, there are at most finitely many $x \in \{0,1\}^n$ such that the following holds: When given input x, the algorithm $R(x)$ prints an $O(n)$-length description of a machine M that runs in space $C\cdot\log(n)$ and prints $f(x)$. Then, for any $S(n) = \Omega(\log n)$ and constant $\varepsilon > 0$,*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{SPACE}[(2+\varepsilon)\cdot S].$$

In the time-bounded regime, deducing extremely efficient (i.e., superfast) derandomization from a strong circuit lower bound is still an open problem, let alone deducing it from hardness for uniform deterministic algorithms; currently, all works require either cryptography, or lower bounds for non-deterministic non-uniform circuits (see, e.g., [DMOZ22, CT21b, CT21a, SV22, CT23]).

Disambiguating nondeterministic logspace. The final question we consider is whether non-deterministic logspace can be made *unambiguous*, in the sense that for every **NL** language, there is a (one-way logspace) verifier that is only convinced by a *unique* witness for every $x \in L$. This is commonly known as the **NL** = **UL** question, and it is the space-bounded analogue of the **NP** vs. **UP** question. The disambiguation task reduces to a derandomization task, and specifically to derandomizing a graph-theoretic variant of the classical isolation lemma by [VV86], where the variant was introduced by Reinhardt and Allender [RA00] (see also [GW96]).

Allender, Reinhardt, and Zhou [ARZ99] showed that if there is a problem in $\mathbf{SPACE}[O(n)]$ hard for non-uniform exponential-sized circuits, then indeed **NL** = **UL**. Very recently, Li, Pyne, and Tell [LPT24] proved an analogous conclusion in a scaled-up regime (i.e., $\mathbf{NSPACE}[n] = \mathbf{USPACE}[O(n)]$) from hardness for *uniform* exponential-sized circuits (where the machine printing the circuit is itself an $O(n)$ space unambiguous machine). In this context too, we deduce the same conclusion from lower bounds against uniform *polynomial-sized* circuits.

Theorem 6. *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that $\mathbf{USPACE}[n]$ is hard for circuits of size n^c with oracle access to $\mathbf{USPACE}[\varepsilon cn]$, where the circuits are uniformly generated by an algorithm that runs in $\mathbf{TISP}[2^{O(n)}, \mathrm{poly}(n)]$ with oracle access to $\mathbf{USPACE}[O(n)]$.*[5] *Then, $\mathbf{NSPACE}[n] \subseteq \mathbf{USPACE}[O_\varepsilon(n)]$.*

Similarly to Theorem 4, the hardness assumption in Theorem 6 represents a near-exponential improvement in the size of the circuits against which we need hardness, at the cost of mildly increasing the space allowed for the uniform machine.

[5] Here too, the input length n to the oracle is the same length as the input to the generating algorithm.

1.3 The Technical Contributions

Our results are based on substantial improvements to the array of technical tools that have been introduced in recent years for studying hardness-vs.-randomness for bounded-space computation. To contextualize this contribution, consider classical constructions of pseudorandom generators based on a hard function f (e.g., the Nisan-Wigderson [NW94] PRG). The analysis of these PRGs is based on a *reconstruction* argument: If an efficient distinguisher is not fooled by the generator (built from f), then an efficient procedure computes f.

Now, let us view both the pseudorandom generator and the reconstruction as a pair of algorithms "of equal status", rather than thinking of the reconstruction as only part of the analysis; similar perspectives have been useful for extractor theory, meta-complexity, learning, and pseudodeterministic algorithms (see, e.g., [TSUZ07, ISW06, CIKK15, Hir23, CLO+23]). Observe that a generator with respect to f is useful for derandomization, whereas the reconstruction procedure computes f, and *at least one* of the two is guaranteed to work (cf., Theorem 1). We will use a function f such that both the output of the generator (when f is hard) and the output of f itself (when f is easy) are useful.

A key point for making this approach work is using both a generator and a reconstruction procedure with low complexity. In recent years, *deterministic* reconstruction procedures have been developed, following Pyne, Raz, and Zhan [PRZ23] (see also [DPT24, LPT24]), in which case both the generator and the reconstruction algorithms are deterministic. Technically, in this work we develop new efficient generators with deterministic reconstruction procedures, as well as deterministic reconstruction procedures for known generators that work in broader contexts than before. Specifically, our results rely on the following technical contributions:

1. **Derandomized D2Ps for PRG+Distinguisher.** All known derandomized reconstruction procedures rely on derandomized transformations of distinguishers to predictors (D2P). Informally, a D2P transformation is a mapping from circuits C into short sequences $P_1, \ldots, P_m$ of circuits, such that if C distinguishes a distribution $\mathbf{D}$ from uniform, then some P_i is a decent next-bit predictor for $\mathbf{D}$. Yao's [Yao82] classical lemma can be thought of as a very general randomized D2P, whereas we are interested in deterministic D2Ps.

   Previously, deterministic D2Ps were known either for read-once branching programs [DPT24] or for specific distinguishers [LPT24]. We develop deterministic D2Ps for *compositions of PRGs with distinguishers*, where both the distinguisher and the PRG may be of various types: Our D2Ps work for compositions of Nisan's [Nis94] PRG with ROBPs, of the Forbes-Kelley [FK18] PRG with AOBPs, and of a PRG by van Melkebeek and Prakriya [vMP19] with a graph-theoretic distinguisher. See Section 2.1.2, Section 2.3, and Section 6.3 for details.

2. **SU Generator with deterministic reconstruction.** Previous deterministic reconstruction procedures were for generators or targeted generators based on the Nisan-Wigderson generator [NW94] (e.g., for the targeted generator of [CT21a] instantiated with [NW94]; see also [PRZ23, DPT24, LPT24]). However, the NW generator is well-known to have suboptimal parameters, and using it in our constructions would not allow us to obtain our results. We thus develop a deterministic low-space reconstruction procedure for the more efficient Shaltiel-Umans [SU05] generator, which we use for our results. See Section 2.1.3 for details.

3. **A new targeted generator.** For the pair of algorithms in Theorem 1, we construct a new targeted generator with a deterministic reconstruction procedure. This generator can be thought of as a variant of the Chen–Tell [CT21a] generator that is particularly suited for space-bounded hardness-vs.-randomness results. The construction is described in Section 2.1.

2 Overview of Proofs

In this section we present high-level overviews of our proofs, aiming to present self-contained descriptions (especially for the proof of Theorem 1). In particular, while describing the proofs we will explain the role of the new complexity-theoretic tools mentioned in Section 1.3 (i.e., the D2P transformations, the reconstructive generator, and the targeted generator for space-bounded settings), but we will present the constructions of these tools in separate subsections. In particular, in Section 2.1 we explain the proof of Theorem 1, in Section 2.2 we explain how to deduce the win-win corollaries, and in Section 2.3 we explain the proofs of results from Section 1.2.

2.1 The Pair of Algorithms

At a high level, our algorithms $\mathcal{A}_1, \mathcal{A}_2$ (for random walk estimation and s-t connectivity respectively) work as follows. Fixing a graph G on n vertices, we consider a reachability bootstrapping system (à la [CT21a]), which is a sequence of n strings ("layers") defined as follows. For $i \in [n]$, the i^{th} layer, denoted $P_i \in \{0,1\}^{n^2}$, is defined as:

$$(P_i)_{s,t} = \mathbb{I}\left[\text{there exists a path from } s \text{ to } t \text{ of length at most } i\right].$$

We observe two critical properties about this system:

1. **Downward self reducibility.** There is a (deterministic) logspace algorithm that, given input $(G,(s,t))$ and query access to P_i, computes $(P_{i+1})_{s,t}$.
2. **Nondeterministic computability.** There is a nondeterministic logspace algorithm that, given (G,i,s,t), computes $(P_i)_{s,t}$.

The algorithm in Item 1 is direct, whereas the algorithm in Item 2 requires the Immerman-Szelepcsényi theorem [Imm88, Sze88] that **coNL** = **NL**. Note that we could use the first algorithm to compute any entry in the bootstrapping system (by repeatedly using downward self-reducibility), but the recursion depth is n, yielding a space-inefficient algorithm. The second algorithm allows us to "shortcut", and compute each entry in the bootstrapping system using nondeterminism.

> We build the pair of algorithms $\mathcal{A}_1, \mathcal{A}_2$ around the following question: Is there an i such that P_i allows us to produce pseudorandom walks on G (using complexity-theoretic tools)? Given such an i, we will estimate random walk probabilities; otherwise, we will solve s-t connectivity.

Specifically, for the purpose of producing pseudorandom walks from P_i, we build a pseudorandom generator GEN with the following properties:

1. **Logspace computability.** Given $P \in \{0,1\}^{n^2}$ and a graph G on n vertices, $\mathsf{GEN}^P(G)$ is computable in logspace. Moreover, the output is either $\widetilde{\mathbf{G}} \in \mathbb{R}^{n \times n}$ or $\perp$, where $\widetilde{\mathbf{G}}$ is a $1/n$-approximation of $\mathbf{G}^n$, and $\mathbf{G}$ is the random walk matrix of G.
2. **Deterministic reconstruction.** There is an algorithm REC running in polynomial time and polylogarithmic space that, given P and G such that $\mathsf{GEN}^P(G) = \perp$, outputs a polylog$(n)$-size (oracle) circuit C such that $C^G(x) = P_x$ for all $x \in [|P|]$.

First we explain how to combine these ingredients to obtain Theorem 1, then go further into the description of the generator. To estimate random walk probabilities in **NL**, we enumerate over i, and use $\mathsf{GEN}(G)$ with P_i to try to produce $\widetilde{\mathbf{G}} \approx \mathbf{G}^n$. When $\mathsf{GEN}(G)$ tries to access entries of P_i,

we answer using the **NL** algorithm from Item 2. This yields an **NL** algorithm $\mathcal{A}_1$ such that if there is an i for which $\mathsf{GEN}^{P_i}(G) \neq \perp$, the algorithm outputs an approximation of n-step walks on G.[6]

Otherwise, it is the case that $\mathsf{GEN}^{P_i}(G) = \perp$ for every i. In this case, we iterate from $i = 1, \ldots, n$, at each stage using REC to build a compressed representation C_i of P_i. This compressed representation is of size $\mathrm{polylog}(n)$, and can be evaluated in space $\mathrm{polylog}(n)$.[7] Once we have a compressed representation C_n of P_n, we can output the transitive closure of G using $\mathrm{polylog}(n)$ space and $\mathrm{poly}(n)$ time. In more detail, in each iteration $i \in [n]$:

1. Assume we have a representation C_{i-1} such that $C_{i-1}^G(s,t) = (P_{i-1})_{s,t}$ for all $(s,t) \in [n]^2$. By Item 1, we can compute $(s,t) \mapsto (P_i)_{s,t}$ using queries to C_{i-1}.

   (In the first iteration $i = 1$ we can compute each entry of P_1 directly in logspace.)

2. By our assumption, $\mathsf{GEN}^{P_i}(G) = \perp$. Hence, we can use the algorithm REC from Item 2 to obtain C_i such that $C_i^G(s,t) = (P_i)_{s,t}$ for all s, t.

3. Finally, delete the representation C_{i-1}, and increment i.

Since each of the n steps takes $\mathrm{polylog}(n)$ space and $\mathrm{poly}(n)$ time, and space is reused across steps, the algorithm $\mathcal{A}_2$ runs in $\mathbf{SC} = \mathbf{TISP}[\mathrm{poly}(n), \mathrm{polylog}(n)]$ as claimed.

Finally, note that both algorithms can detect failure. Specifically, the **NL** algorithm $\mathcal{A}_1$ outputs $\perp$ if no i allowed it to produce $\widetilde{\mathbf{G}}$ (i.e., if $\mathsf{GEN}(G)$ outputs $\perp$ with all P_i). Similarly, the **SC** algorithm $\mathcal{A}_2$ can check at each iteration i that C_i^G computes P_i (otherwise, it outputs $\perp$).

Outline of Technical Description. In Section 2.1.1, we give an overview of the construction of GEN. Then, in Sections 2.1.2 and 2.1.3, we describe constructions of two key technical components that are necessary for the construction of GEN.

2.1.1 A walk generator with deterministic reconstruction

The construction of GEN is based on the hardness vs. randomness paradigm. As explained in Section 1.3, this paradigm yields a pair of algorithms GEN and REC such that for every string P, either GEN produces pseudorandomness from P or REC efficiently computes P (in this case, REC produces a small description of P). The pseudorandomness that we need in our setting is very specific: we just need to approximate walk probabilities on a graph.

A first attempt might be to instantiate a known PRG, such as NW [NW94, IW97], with the truth table $f = P_i$, and use the output of the generator to take random walks on G. A line of recent work [CH22, PRZ23, DT23, DPT24, LPT24] developed deterministic and space-efficient reconstruction procedures for NW that work in our setting (i.e., when using the output to approximate walks), but there is a fundamental issue: If we use NW to produce n pseudorandom bits (for an n-step walk), then we can only guarantee that when NW fails, the reconstruction compresses f to size n^c (for some constant $c > 1$), which in our case would not compress the reachability matrix at all.

Indeed, we need a generator that will produce n-step pseudorandom walks, such that failure of this generator allows us to compress f to size $\mathrm{polylog}(n)$. This is known to be impossible when

[6] To ensure the output is consistent for a fixed graph, the algorithm tries $i = 1, \ldots,$ in sequence and uses the first layer that produces an output.

[7] To be more accurate, the algorithm needs to evaluate C_i^G rather than C_i. But since G is given to the algorithm as input, whenever C_i queries G, the algorithm can answer the query in logspace.

using standard black-box techniques to produce n pseudorandom bits,[8] but since we only care about producing pseudorandom walks for a given graph (rather than arbitrary pseudorandom bits), we are able to bypass this barrier using two technical components.

Technical component 1: Using an unconditional "outer" PRG. Instead of producing n-step walks, we use the PRG to produce a polylog(n)-bit seed for the pseudorandom generator NIS of Nisan [Nis91], which stretches its seed to an n-step pseudorandom walk.[9] Since our PRG now only needs to output polylog(n) bits, when it fails we can compress f to size polylog(n).

However, this breaks a key part of the argument. Specifically, the reconstruction REC for our PRG relies on a transformation of a *distinguisher* (for the output of the PRG) into a *next-bit-predictor*, à la Yao [Yao82], denoted D2P. Since we need REC to be deterministic, we need the D2P to be deterministic. However, previous works [CH22, PRZ23, DPT24, LPT24] constructed D2P transformations only for certain classes of distinguishers, and the distinguisher *"does the PRG output a seed for* NIS *that causes it to produce good pseudorandom walks?"* is not in these classes.

Thus, we develop a new deterministic D2P for the latter distinguisher. In fact, this is part of a broader contribution of this work, in which we develop deterministic D2Ps for various types of distinguishers of the form "composition of a PRG with a weak distinguisher". Our D2P for the current specific distinguisher is described in Section 2.1.2.

Technical component 2: A better "inner" PRG. A well-known limitation of NW is that when producing polylog(n) output bits and compressing f to size polylog(n), its seed is polylogarithmic rather than logarithmic. Thus, it would not have the properties that we need from GEN (i.e., logspace and polytime computability). Instead of using NW, we use a generator that does not suffer from this limitation, namely the Shaltiel–Umans [SU05] generator SU.

However, now another key part in the argument breaks. Recent works developed deterministic logspace reconstruction algorithms for NW, but no such algorithms are known for SU. In fact, only very recently Chen *et al.* [CLO+23] showed a setting in which (a modification of) SU has reconstruction that can be computed uniformly (rather than by non-uniform circuits), and their algorithm is neither space-efficient nor randomness-efficient. Thus, we develop a new reconstruction procedure, which simultaneously achieves both these goals. This is described in Section 2.1.3.

By combining the two foregoing technical components, we obtain GEN = SU such that GEN and REC have the properties needed to construct the pair of algorithms described in Section 2.1.

2.1.2 A D2P for Nisan's generator composed with estimating random walks

Let us recall the definitions of distinguishers, predictors, and of D2P transformations (the latter were formally introduced in [DPT24] and studied more generally in [LPT24]).

Definition 2.1 (distinguisher)**.** We say that $C\colon \{0,1\}^n \to \{0,1\}$ is an ε-distinguisher for a distribution $\mathbf{D}$ over $\{0,1\}^n$ if $\left|\mathbb{E}[C(\mathbf{U}_n)] - \mathbb{E}[C(\mathbf{D})]\right| \geq \varepsilon$, where $\mathbf{U}_n$ is the uniform distribution.

[8] To be precise, whenever using the standard hybrid argument (or, more generally, a distinguish-to-predict transform for arbitrary distinguishers) and black-box hardness amplification to produce n pseudorandom bits, an overhead of poly(n) in compression is unavoidable; see [GSV18, SV22] and [LPT24, Appendix B].

[9] In this simplified description, we consider NIS as an algorithm that takes as input a random seed and produces a set of random walks (and for most seeds, the uniform distribution over the set of walks is pseudorandom). This idea was also used to reduce the catalytic space complexity of producing random walks [Pyn24].

Definition 2.2 (next-bit predictor)**.** For $i \in [n]$, we say that $P\colon \{0,1\}^{i-1} \to \{0,1\}$ is a δ-next-bit-predictor for a distribution $\mathbf{D}$ over $\{0,1\}^n$ if $\Pr_{x \leftarrow \mathbf{D}}[P(x_{<i}) = x_i] \geq \frac{1}{2} + \delta$.

Definition 2.3 (D2P, simplified; see Definition 3.2)**.** An algorithm A is a distinguish to predict (D2P) transformation for a class $\mathcal{C}$ if A gets as input a description of a circuit $C\colon \{0,1\}^n \to \{0,1\}$ from $\mathcal{C}$, and prints a list of circuits $P_1, ..., P_m\colon \{0,1\}^\star \to \{0,1\}$ such that for *every* distribution $\mathbf{D}$ over $\{0,1\}^n$ the following holds. If C is a $(1/3)$-distinguisher for $\mathbf{D}$, then there is an $i \in [m]$ such that P_i is an $(1/O(n))$-predictor for $\mathbf{D}$.

Several prior works [Nis94, CH22, GRZ23, PRZ23, DPT24] yield deterministic D2P transformations for the "random walk" distinguisher. In more detail, this distinguisher can be modeled as a read-once branching program (ROBP), and deterministic D2Ps for ROBPs are known.

Now, recall that Nisan's PRG NIS chooses at random $\ell = \log n$ hash functions $\vec{h} = (h_1, \ldots, h_\ell)$, each over $O(\log n)$ bits, and for every graph G, for almost all collections of hash functions, the generator $\mathsf{NIS}_{\vec{h}}$ produces a pseudorandom distribution of random walks for G. Our distinguisher is therefore

$$T_G(\vec{h}) = \mathbb{I}\left[\mathrm{NIS}_{\vec{h}} \text{ produces good random walks for } G\right],$$

and this is not an ROBP (as the Nisan PRG reads each hash function repeatedly even to produce a single output). We overcome this issue by constructing a D2P transformation for this distinguisher:

Theorem 2.4 (informal; see Section 4.1)**.** *There is a deterministic logspace D2P transformation for T_G. Moreover, each candidate predictor is evaluable in logspace, given access to G.*

Our proof uses a reduction from the recent work of Li, Pyne, and Tell [LPT24]. They show that producing a D2P transformation for a distinguisher T reduces to solving a problem called "prefix-CAPP" (PCAPP) for T.[10] In particular, we say that a logspace machine solves PCAPP for T if given $T\colon \{0,1\}^n \to \{0,1\}$ and $x \in \{0,1\}^{\leq n}$, the machine estimates $\mathbb{E}_z[T(x \circ z)]$ to within error $1/n^2$.

Theorem 2.5 ([LPT24], see Lemma 3.5)**.** *Suppose there is a logspace machine that solves* PCAPP *for T. Then, there is a logspace computable D2P transformation for T.*

Naively, solving such a PCAPP problem may seem as hard as solving CAPP directly for T (which itself in general is as hard as derandomization). However, we exploit the structure of T to solve PCAPP more efficiently. In [LPT24] they observed that this is possible if the distinguisher obeys a certain *polarization* property, which in our case is as follows. For every G, and prefix of hash functions $(h_1, \ldots, h_i)$, the following dichotomy occurs:

1. *There is a $j \leq i$ such that h_j is a "bad" hash function.* In this case, for every suffix z, $\mathbb{E}_z[T_G(h_1, \ldots, h_i, z)] = 0$.
2. *There is no $j \leq i$ such that h_j is a bad hash function.* In this case, $\mathbb{E}_z[T_G(h_1, \ldots, h_i, z)] \approx 1$.

We show that (a slight modification of) the generator NIS indeed obeys this polarization property. Hence, when considering the output distribution of this (modified) generator, we can solve PCAPP in a simple, deterministic way: we only need to test each of the hash functions in the given prefix (i.e., rather than estimate T on a distribution of suffixes). By the reduction of D2P to PCAPP from [LPT24], we obtain an efficient D2P transformation for T_G.

[10] CAPP is short for Circuit Approximation Probability Problem [KRC00], the problem of estimating the probability of acceptance of a given circuit C to within a fixed additive error. In a "prefix-CAPP" problem, we are given a prefix x along with a device T, and wish to approximate the probability of acceptance of $T(x \circ y)$ over random suffixes y.

2.1.3 A generator with uniform near-deterministic logspace reconstruction

The generator SU maps $f \in \{0,1\}^N$ to a list of strings in $\{0,1\}^M$, which are hopefully pseudorandom. It is coupled with an efficient reconstruction algorithm RSU that converts any next-bit-predictor P for the list of M-bit strings into a circuit C_f^P of size $\mathrm{poly}(M) \ll N$ that computes f. In our setting $N = n^2$ and $M = \mathrm{polylog}(n)$, and it is crucial that RSU is a small-space machine that uses only $O(\log N)$ random coins (so that we can enumerate).

A formal statement appears in Theorem 5.1, and we now describe some of the ideas in our modification, at a high-level. The generator SU, which we do not change, arithmetizes its input f as a low-degree polynomial $\hat{f}\colon \mathbb{F}_q^v \to \mathbb{F}_q$, and outputs evaluations of $\hat{f}$ on "lines" going in a certain direction A in $\mathbb{F}_q^v$.[11] A simplified version of the reconstruction is as follows: Choose a random low-degree curve $C\colon \mathbb{F}_q \to \mathbb{F}_q^v$, and query $\hat{f}$ at the $m-1$ "preceding" curves going back from C in direction A^{-1} (i.e., query $\hat{f}$ at all points on the curves $A^{-i} \cdot C$ for $i \in [m-1]$). This curve C and queried "starting points" define a circuit F, which computes $\hat{f}$. Specifically, when given $\vec{z}$, the circuit F starts from $C_0 = C$ and repeatedly uses the next-element-predictor – which predict in "direction" A – to predict the next curve $A \cdot C_{i+1}$, until reaching a curve that contains $\vec{z}$. (This simplified description hides many details, among them the fact that D works in several "strides" of the form A^i for $i = q, ..., q^{v-1}$, and the fact that C actually uses two interleaved curves and relies on a list-decoding algorithm at each step.)

Let us briefly explain how we make this reconstruction randomness-efficient. (Making the reconstruction space-efficient is relatively easier, relying on the efficiency of many of its components as well as on ideas of Doron and Tell [DT23].)

Randomness-efficient samplers. Following Pyne, Raz, and Zhan [PRZ23], we use randomness-efficient samplers to reduce the randomness complexity of various parts of the reconstruction. The underlying observation is that many of the components in the reconstruction repeat a single procedure that uses $O(\log N)$ coins multiple times (and take, for example, an OR, or the majority vote). Instead of repeating the procedure with independent coins, we can use a sampler with $O(\log N)$ coins to output a sample such that (w.h.p.) the procedure behaves on the sample approximately the same as on a uniformly chosen sample.

Pseudorandom curves, and reusing randomness for defining points. The procedure relies on the random low-degree curve having sufficiently strong sampling properties. First, it needs the curve to be a good sampler (i.e., for any subset $T \subseteq \mathbb{F}_q^v$, the points on the curve sample T approximately correctly, with high probability). A natural idea is to replace a random curve with a *curve sampler*, which pseudorandmly outputs a curve with the needed sampling properties. Indeed such a sampler was designed by Guo [Guo13] (following [TU06]) for this particular purpose – and in fact with our parameter regime in mind.

Secondly, when considering the $\mathsf{defining\ points}$ for the ℓ-degree curve, which are the points $t_1, ..., t_\ell \in \mathbb{F}_q$ such that we interpolate the curve according to certain values on $t_1, ..., t_\ell$, the procedure splits these points into $O(\log N)$ blocks, and needs the points inside each block to be good samplers (in $\mathbb{F}_q$).[12] A naive approach is to choose the points in each block using a sampler, but this seemingly requires too much randomness (i.e., $O(\log N)$ coins, times the sampler's randomness

[11] To be more accurate, consider a matrix $A \in \mathbb{F}_q^{v \times v}$ representing multiplication by a primitive element in $\mathbb{F}_{q^v}$. Then, the generator chooses a random $\vec{x} \in \mathbb{F}_q^v$ and outputs the evaluations of $\hat{f}$ at the points $\vec{x}, A \cdot \vec{x}, A^2 \cdot \vec{x}, ..., A^{m-1} \cdot \vec{x}$.

[12] This sampling property of the defining points of the curve C is used to argue that, with high probability, the predictor succeeds in predicting sufficiently many points on C (and on each $A^i \cdot C$).

complexity). However, we show that the same randomness can be reused across blocks, since the analysis boils down to a union-bound over events that each depend on a single block.

Pseudorandom interleaving. The last part of our modification is more subtle. Loosely speaking, in [SU05] they actually use two low-degree curves C_1 and C_2, where C_1 is a random curve and C_2 is obtained by "shifting" the values of C_1 on some of the defining points, by a small number $O(\log N)$ of predetermined shift values.[13] In their argument, both C_1 and C_2 have sufficient sampling properties, since the marginal distribution over each of the curves is that of a random low-degree curve. However, in our argument, C_2 is obtained by applying a sequence of predetermined shifts to the values of a curve sampler C_1 on the defining points (and then interpolating), and it is not clear that this operation preserves the sampling properties of C_1.

If C_2 would have been obtained by applying shifts to *all* of the points of C_1 (rather than applying them to the defining points and then interpolating), then we would be able to prove that C_2 is indeed a sampler. Of course, we cannot enforce that all of the points of C_2 will be various shifts of C_1, since that is not necessarily a low-degree curve. To get around this, we partition $\mathbb{F}_q$ into a small number $O(\log N)$ of large subfields, and for each subfield, we use a sampler to choose defining points in the subfield, and define C_2 using an appropriate shift of C_1 on these defining points. As above, we reuse randomness for the sampler across subfields.[14] Since C_2 is a shift of C_1 on a pseudorandom set of points within each subfield, we can argue that C_2 behaves sufficiently similar to a shift of C_1 on the entire subfield. Further details appear in Section 5.4 and in Proposition 5.20.

2.2 Tight Win-Win Results in Space-Bounded Complexity

We now explain how to obtain the win-win results in space complexity, based on (the proof of) Theorem 1. For Theorem 2, we first modify the pair of algorithms. Rather than attempting to compute reachability and random walks on the same graph (equivalently, on two graphs of comparable size), we instead take a small graph G_1 (of size $2^{\log^{1/2+\varepsilon/2} n}$), and a large graph G_2, and attempt to either compute reachability on G_1, or estimate random walk probabilities on G_2. Also, instead of computing the reachability bootstrapping system in **NL**, we use Savitch's Theorem [Sav70] to compute the system deterministically. This yields the following pair of algorithms:

Theorem 2.6 (informal; see Theorem 6.7)**.** *For every $\varepsilon > 0$, there are algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for every pair of graphs G_1 on $2^{\log^{1/2+\varepsilon/2} n}$ vertices, and G_2 on n vertices, at least one of the following holds:*

- *$\mathcal{A}_1(G_1, G_2)$ computes s-t connectivity in G_1 in* **SC**.
- *$\mathcal{A}_2(G_1, G_2)$ estimates length-n random walk probabilities in G_2 in* $\mathbf{SPACE}[\log^{1+\varepsilon} n]$.

Moreover, both algorithms report if they fail to compute the desired answer, and do not exceed their resource bounds in any case.

Indeed, in the proof of Theorem 2.6, the reachability bootstrapping system can be computed in nondeterministic space $O(\log^{1/2+\varepsilon/2} n)$, and hence in *deterministic* space $O(\log^{1+\varepsilon} n)$ by Savitch's Theorem [Sav70]. As such, in the case that the reachability bootstrapping system does contain a

[13] That is, if C_1 is defined by a small number of conditions of the form "$C_1(t) = \vec{z}_t$" (for a small number of $t \in \mathbb{F}_q$ and $\vec{z}_t \in \mathbb{F}_q^v$), then C_2 is defined by the conditions "$C_2(t) = A_t \cdot \vec{z}_t$", where A_t is an invertible matrix in $\mathbb{F}_q^v \times \mathbb{F}_q^v$.

[14] In fact, we set things up so that the $O(\log N)$ subfields we use for pseudorandom interleaving are also exactly the $O(\log N)$ blocks mentioned above (when discussing pseudorandom curves).

hard truth table, we can compute this truth table (and hence compute random walks on G_2) in space $O(\log^{1+\varepsilon} n)$, obtaining a small overhead even for deterministic derandomization.

We use the pair of algorithms from Theorem 2.6 to prove Theorem 2. Following the approach of [DPT24, LPT24], we fix a $\mathbf{BPSPACE}[n]$ machine $\mathcal{B}$ and a $\mathbf{NSPACE}[n^{1/2+\varepsilon/2}]$ machine $\mathcal{N}$. For each pair $(x, y) \in \{0,1\}^n \times \{0,1\}^n$, consider the following two graphs:

$$G_1(y) = \text{the configuration graph of } \mathcal{N}(y), \quad G_2(x) = \text{the configuration graph of } \mathcal{B}(x) \ .$$

Let us try and simulate $\mathcal{B}$ in $\mathbf{SPACE}[n^{1+\varepsilon}]$ (if we fail, we will show another algorithm that simulates $\mathcal{N}$ in $\mathbf{TISP}[2^m, \mathrm{poly}(m)]$). Our algorithm gets $x \in \{0,1\}^n$ and enumerates over all $y \in \{0,1\}^n$ (which it can, since it is allowed to use super-linear space). It then runs $\mathcal{A}_2$ on the pair of graphs $G_1(y), G_2(x)$; if $\mathcal{A}_2$ outputs an estimation of random walks on $G_2(x)$, we are done, and otherwise we continue to the next y. The point is that one of two things happened: Either for every x there is y such that this algorithm succeeds, in which case we simulated $\mathcal{B}$ in $\mathbf{SPACE}[n^{1+\varepsilon}]$ on this input length; or there exists x such that for every y this algorithm fails. In the latter case, a symmetric argument shows that for all $y \in \{0,1\}^n$ we can simulate $\mathcal{A}_1$ in $\mathbf{TISP}[2^m, \mathrm{poly}(m)]$ (i.e., given y, we enumerate over all x and simulate $\mathcal{A}_1$ with the two graphs). For further details see Section 6.2, and in particular, another win-win result appears in Theorem 6.10.

2.3 Derandomization from Very Weak Hardness

The basic idea behind all of our results that deduce derandomization from very weak hardness (i.e., Theorem 4, Theorem 6 and Theorem 2.7) is the same. Let us describe the idea in general terms, and then later focus on describing one particular result in more technical detail.

To demonstrate the idea, consider trying to derandomize $\mathbf{BPSPACE}[O(n)] \subseteq \mathbf{SPACE}[O(n)]$. We instantiate a generator with a hard problem $L \in \mathbf{SPACE}[O(n)]$, say, the Shaltiel–Umans generator SU from Section 2.1.3. Now, if the derandomization fails on some input, then we can compress the hard truth-table, as follows. We enumerate over inputs $x \in \{0,1\}^n$ (indeed, we can do this since we are working in a scaled-up regime of linear space) and run the reconstruction algorithm. This algorithm is deterministic, and whenever the derandomization fails at x, the reconstruction manages to compress the truth-table of $L_{O(n)}$. (For simplicity, we ignore for a moment the distinguisher that is not fooled by SU and whose description is part of the compressed version of $L_{O(n)}$.)

The key question is what is the *size* of the compressed version of $L_{O(n)}$. For this question, a main bottleneck is the number of output bits that we ask SU to output; loosely speaking, if it outputs M bits, then the compressed version will be of size $\mathrm{poly}(M)$.[15] In settings concerning derandomization in polynomial time or logarithmic space, we have $M = N^{\Omega(1)}$, and thus $\mathrm{poly}(M) = N^{\Theta(1)}$. For our results, we are interesting in obtaining a compressed representation of size only $\mathrm{polylog}(N)$.

The key: Unconditional PRGs and corresponding D2Ps. The way to achieve this goal will be similar to an idea from the proof of Theorem 1. In all of our settings, we consider relatively weak distinguishers, for which unconditional PRGs are known. For example, when derandomizing $\mathbf{BPSPACE}[O(n)]$, the distinguisher D is an ROBP, and we can compose it with Nisan's [Nis91] PRG to obtain a distinguisher $D \circ \mathsf{NIS}$ over $M = \mathrm{polylog}(N)$ bits.

The main challenge is that we now need to develop derandomized D2P transformations for distinguishers of the form "compose a weak distinguisher with an unconditional PRG". Indeed, one such D2P was presented in Section 4.1, for a composition of ROBPs with NIS. To further

[15] This is since the reconstruction overhead is affected by the prediction advantage, and the prediction advantage is at most $1/M$ whenever using a hybrid argument (or a deterministic D2P; see [LPT24, Appendix B]).

demonstrate out approach, we now describe another derandomized D2P, for the composition of an AOBP with (a modified version of) the Forbes-Kelley [FK18] PRG. The AOBP distinguisher – which is an *any-order* branching program, defined in [CLTW23] – comes up when constructing derandomization algorithms with minimal memory overhead (and when using an idea from [DT23]; see Lemma 6.15), and the FK PRG fools such distinguishers with polylogarithmic seed.

2.3.1 Derandomized D2P for $\mathsf{AOBP} \circ \mathsf{FK}$

Consider an AOBP denoted A and the Forbes-Kelley PRG FK. Note that even if A would have been an ROBP, the composition $A \circ \mathsf{FK}$ is not an ROBP, and thus we cannot use the known D2P transformations for ROBPs (e.g., from [DPT24]) for this composition.

We construct a derandomized D2P transform for a modified version of the Forbes-Kelley generator, where the modification facilitates the D2P transform. For our goal of minimal-memory overhead it will be crucial that the modified version of FK remains strongly explicit (as is the original generator), but we can afford a seed length that is n^{ε} (rather than $\mathrm{polylog}(n)$). We give an informal statement of the result here:

Theorem 2.7 (Forbes–Kelley D2P, informal)**.** *There is a generator* $\mathsf{FK}\colon \{0,1\}^{n^{\varepsilon}} \to \{0,1\}^n$ *with the following properties.*

1. ***Strong Explicitness.*** *The map* $(x, j) \to \mathsf{FK}(x)_j$ *is computable in space* $O(\varepsilon \log n)$ *with catalytic access to* j*.*[16]
2. ***Fooling.*** *The generator fools AOBPs of size* n *to error* $1/n$*.*
3. ***White-Box D2P.*** *There is a white-box D2P transform that can be computed in time* $\mathrm{poly}(n)$ *and space* $O(n^{\varepsilon})$*, where each predictor can be evaluated in space* $\log(n) + O(\varepsilon \log n)$*.*[17]

Our actual construction makes several changes to the Forbes–Kelley generator, but the idea is the following. The generator is constructed as a sequence of *random restrictions*, each of which eliminate some variables while approximately preserving the expectation of the branching program A. We think of the generator's input as $(A_1, B_1, \ldots, A_\ell, B_\ell)$, where each A_i, B_i is the output of a k-wise independent generator over $\{0,1\}^n$. We then define $\mathsf{FK}_{\ell+1} = 0^n$, and

$$\mathsf{FK}_i = A_i \oplus B_i \wedge \mathsf{FK}_{i+1}$$

In particular, for $j \in [n]$ where $(B_i)_j = 0$, we say a variable has been eliminated by level i, and further levels of the generator do not affect the output of the generator on that bit. Recall that by [LPT24], to produce a D2P transform, it suffices to solve prefix-CAPP, which in this case is the following:

Question 2.8. Given an AOBP A and $(\vec{a}, \vec{b}) = (A_1, B_1, \ldots, A_{i-1}, B_{i-1})$, estimate

$$\mathop{\mathbb{E}}_{\vec{a}', \vec{b}'}[A(\mathsf{FK}(\vec{a}, \vec{b}, \vec{a}', \vec{b}')].$$

[16] The algorithm is given access to a special read-write tape, initialized to (the binary representation of) j. When the algorithm halts and returns $\mathsf{FK}(x)_j$, the tape must be restored to that initial configuration.

[17] For technical reasons we construct a white-box Yao derandomization [LPT24], where we are given access to the distribution $\mathbf{D}$ that does not fool $A \circ \mathsf{FK}$ and construct a predictor for this distribution.

To see how the prefix affects the problem, consider the branching program $A_{\vec{a},\vec{b}}$ wherein every variable that has been eliminated is simply fixed to the value output by the generator at that bit. Thus, we are now being asked to estimate

$$\mathbb{E}[A_{\vec{a},\vec{b}}(z \oplus \mathsf{FK}_i(\mathbf{U}))].$$

where z is a fixed vector that accounts for the prior levels of the generator.

The key observation is that the Forbes-Kelley generator *fools branching programs*, and so this expectation should itself be close to $\mathbb{E}[A_{\vec{a},\vec{b}}(\mathbf{U})]$, i.e., filling in all non-eliminated variables with independent true randomness. Estimating this quantity is easily seen to be in $\mathbf{BPL} \subseteq \mathbf{SC}$, and so we obtain a $\mathbf{SC}$-computable D2P transform with this complexity.

We remark that our actual construction is substantially different from the above due to the following technical issues: First, we cannot afford $k = O(\log n)$-wise independent restrictions, as at several steps in our argument we must enumerate over all seeds in a single level of a generator, which would take time $2^{\log^2 n} \gg n$. Because of this, we can only afford $O(1/\varepsilon)$-wise independence, which requires a generator with n^ε levels, and a much smaller restriction probability.

The second and larger issue is that it is not actually true that for *every* prefix of a seed to the Forbes-Kelley generator, the output of the generator on a random suffix of the generator is approximately the same as filling in bits uniformly. For example, consider a pathological prefix that eliminates far fewer variables than it should; then, a random suffix of a seed for the generator will not eliminate all the variables (with high probability), in which case the generator fills in many bits with $\mathsf{FK}_{\ell+1} = 0^n$. (Needless to say, this is very far from uniform.)

Our solution is to construct an *auxiliary* D2P transform such that, if many prefixes are in deficient in this way, then we can solve PCAPP on these prefixes very efficiently. This modification requires changing the final level of the generator to behave differently when few variables are left alive. By itself, this change would destroy strong explicitness, but we are able to use ideas from catalytic computation to obtain a strongly explicit PRG with *catalytic access* to the input (which suffices for the minimal memory overhead application, as explained in Section 6.4). The interested reader is referred to Section 4.3 for further details about the D2P.

3 Preliminaries

Given $x \in \{0,1\}^n$, let $x_{<i} = x_{1\ldots i-1}$ and $x_{>i} = x_{i+1\ldots n}$, and let $x_{\leq i}$ and $x_{\geq i}$ be defined analogously. For convenience we denote $x_{<1}$ and $x_{>n}$ as the empty string.

Given a set S, let $\mathbf{U}_S$ be the uniform distribution over the set S, and for $n \in \mathbb{N}$ we denote $\mathbf{U}_n = U_{\{0,1\}^n}$. Also, drawing $x \in S$ is a shorthand for $x \leftarrow \mathbf{U}_S$.

Graphs. For a directed graph G on n vertices, the transitive closure of G is the $n \times n$ matrix where entry (i,j) is 1 if and only if there is a path from i to j. The random walk matrix, which we denote $\mathbf{G}$, is the matrix where (i,j) is the probability of transitioning from vertex i to vertex j in one step. Our matrix norm (which we will use mainly for transition matrices of graphs) will be the induced ℓ_∞-norm on matrices, namely $\|A\| = \max_{i\in[n]} \left|\sum_{j\in[n]} A_{i,j}\right|$.

3.1 Distinguish-To-Predict and Prefix-CAPP

We say that a distribution $\mathbf{D}$ ε-fools a function $f\colon \{0,1\}^n \to \{0,1\}$ if $|\mathbb{E}[f(\mathbf{D})] - \mathbb{E}[f(\mathbf{U}_n)]| \leq \varepsilon$, and if $|\mathbb{E}[f(\mathbf{D})] - \mathbb{E}[f(\mathbf{U}_n)]| \geq \varepsilon$ we say that f is an ε-distinguisher for $\mathbf{D}$. We say that $\mathsf{G}\colon \{0,1\}^s \to \{0,1\}^n$ is an ε-pseudorandom generator (PRG) for a class of functions $\mathcal{F}\colon \{0,1\}^n \to \{0,1\}$ if for every

$f \in \mathcal{F}$, $\mathsf{PRG}(\mathbf{U}_s)$ ε-fools f. We say that G is an ε-hitting set generator (HSG) if for every $f \in \mathcal{F}$ such that $\mathbb{E}[f(\mathbf{U}_n)] \geq \varepsilon$, there exists $z \in \{0,1\}^s$ such that $f(\mathsf{G}(z)) = 1$.

As explained in Section 1.3 and in Section 2, we will be interested in deterministic transformations of distinguishers to predictors. Let us define this notion, following [DPT24, LPT24]:

Definition 3.1 (predictor)**.** We say that $P\colon \{0,1\}^i \to \{0,1\}$ is an ε-next-bit predictor (resp. ε-previous-bit predictor) for a distribution D over $\{0,1\}^n$ if $\Pr_{x \leftarrow D}[P(x_{\leq i}) = x_{i+1}] \geq 1/2 + \varepsilon$ (resp. $\Pr_{x \leftarrow D}[P(x_{>n-i}) = x_{n-i}] \geq 1/2 + \varepsilon$).

Definition 3.2 (D2P, formal)**.** For a circuit $C\colon \{0,1\}^n \to \{0,1\}$, we say a collection of circuits $\mathcal{P}\colon \{0,1\}^{<n} \to \{0,1\}$ is an α-distinguish to δ-predict (D2P) transformation for C if the following holds. For every distribution $\mathbf{D}$ of size at most m that does not δ-fool C, there is $P \in \mathcal{P}$ such that P is an α-predictor for $\mathbf{D}$.

Yao [Yao82] showed what can be thought of as a *randomized* D2P for general circuits. Let us recall his result with a formal statement.

Theorem 3.3 ([Yao82])**.** *Let $C\colon \{0,1\}^n \to \{0,1\}$ be an arbitrary function and $\mathbf{D}$ be an arbitrary distribution that does not δ-fool C. Then there is $i \in [n]$ and $\sigma \in \{0,1\}^2$ such that:*

1. *Letting $P_{z,\sigma,i}$ be defined as $P_{z,\sigma,i}(x_{<i}) = C(x_{<i} \circ \sigma_1 \circ z) \oplus \sigma_2$, we have*

$$\mathop{\mathbb{E}}_{z \leftarrow \mathbf{U}_{n-i}} [\Pr[P_{z,\sigma,i}(\mathbf{D}_{<i}) = \mathbf{D}_i 1]] \geq \frac{\delta}{n}.$$

2. *Moreover, we have*

$$|\mathbb{E}[C(\mathbf{D}_{<i} \circ \mathbf{U}_{n-i+1})] - \mathbb{E}[C(\mathbf{D}_{\leq i} \circ \mathbf{U}_{n-i})]| \geq \frac{\delta}{n}.$$

Moreover (by the reverse Markov's inequality), on at least a $2\delta/(3n-\delta)$ fraction of $z \in \{0,1\}^{n-i}$, we have $\Pr[P_{z,\sigma,i}(\mathbf{D}_{<i}) = \mathbf{D}_i 1] \geq \delta/3n$.

Deterministic D2P via PCAPP. We will use the deterministic reduction from [LPT24] of producing a D2P to solving a problem called PCAPP, where the reduction works instance-wise (i.e., for every fixed circuit C). Let us define PCAPP and state their result:

Definition 3.4. We say a machine $E\colon \{0,1\}^{\leq n} \to \mathbb{R}$ is an ε-prefix-CAPP (PCAPP) algorithm for $C\colon \{0,1\}^n \to \{0,1\}$ if for every $x \in \{0,1\}^{\leq n}$ we have

$$\left| E(C,x) - \mathop{\mathbb{E}}_{r}[C(x \circ r)] \right| \leq \varepsilon.$$

Lemma 3.5 ([LPT24] Lemma 4.6)**.** *Fix $\delta > 0$ and an arbitrary circuit $C\colon \{0,1\}^n \to \{0,1\}$ and function E that is a $(\delta/3n)$-PCAPP algorithm for C. Let $\alpha = \delta/3n$ and $A = \lceil 1/\alpha \rceil$. Then the following is a δ-distinguish to α-predict transformation for C:*

$$\mathcal{P} \stackrel{def}{=} \{P_{\tau,\sigma,i} : i \in [n], \sigma \in \{0,1\}^2, \tau \in \{0,\ldots,A\}\}$$

where

$$P_{\tau,\sigma,i}(x_{<i}) = \mathbb{I}\left[\frac{\tau}{A} \leq E(C, x_{<i} \circ \sigma_1)\right] \oplus \sigma_2.$$

3.2 Space-Bounded Computation

We use the standard model of space-bounded computation (see, e.g., [Gol08, Section 5] or [AB09, Section 4]). In this paper we say that a language is in $\mathbf{SPACE}[s(n)]$ if it is accepted by a machine with space complexity $s(n)$ on inputs of length n, and we stress that we do not allow a linear slack in the space complexity; that is, the space complexity is bounded by $s(n)$ exactly, rather than by $O(s(n))$. We define the space complexity of computing functions analogously.

A *probabilistic* space-bounded machine is similar to the deterministic machine except that it can also toss random coins. As usual, we require a space-$s(n)$ probabilistic machine to always halt within $2^{s'(n)}$ steps, where $s'(n) = s(n) + O(\log s(n)) + \log n$ is the number of possible configurations.[18] Recall that this runtime bound holds wlog for halting space-$s(n)$ *deterministic* machines.

We also recall that $\mathbf{TISP}[t(n), s(n)]$ is the set of languages accepted by a (deterministic) machine that runs in time t and space s, and that $\mathbf{SC} = \mathbf{TISP}[\mathrm{poly}(n), \mathrm{polylog}(n)]$.

Composition of space-bounded algorithms. We recall, and freely use, the standard result on composition of space-bounded algorithms.

Proposition 3.6 ([Gol08], Lemma 5.2)**.** *Let $f_1, f_2 \colon \{0,1\}^\star \to \{0,1\}^\star$ be functions that are computable in space $s_1, s_2 \colon \mathbb{N} \to \mathbb{N}$. Then, $f_2 \circ f_1 \colon \{0,1\}^\star \to \{0,1\}^\star$ can be computed in space*

$$s(n) = s_2(\ell_1(n)) + s_1(n) + O(\log(\ell_1(n))) + O(\log(s_2(\ell_1(n)) + s_1(n)))$$

where $\ell_1(n)$ is a bound on the output length of f_1 (i.e., the cells used on the output tape) on inputs of length n.

3.2.1 Unambiguous space-bounded computation

Definition 3.7 (unambiguous space)**.** A language L is in unambiguous non-deterministic space $S(n)$, denoted as $\mathbf{USPACE}[S(n)]$, if there is a nondeterministic space-S machine $M(x,y) \in \{0,1,\perp\}$ such that for every $x \in L$, there is exactly one witness y such that $M(x,y)$ accepts, and for every $x \notin L$ and every witness y, $M(x,y)$ rejects. We let $\mathbf{UL} = \cup_c \mathbf{USPACE}[c \cdot \log n]$.

We can also compute a *function*, rather than accept or reject, in unambiguous non-deterministic space. We say that $f \colon \{0,1\}^\star \to \{0,1\}^\star$ is computable in unambiguous non-deterministic space $S(n)$ if there exists a non-deterministic space-S machine M such that for any x there exists exactly one witness y for which $M(x,y) = f(x)$, whereas for any other y, $M(x,y) = \perp$. This can be seen as the search analogue of $\mathbf{USPACE}[S(n)] \cap \mathbf{coUSPACE}[S(n)]$. Importantly, we can space-efficiently compose, as in Proposition 3.6, functions that are computable in unambigious non-deterministic space.

3.2.2 Branching Programs

We recall two models of read-once branching program: read-once branching programs (also known as standard-order branching programs), and (read-once) adaptive order branching programs.

Definition 3.8 (ROBP)**.** A read-once branching program (ROBP) B of width w and length n is specified by an initial state $v_{st} \in [w]$, an accepting state $v_{ac} \in [w]$ and a sequence of transition

[18] The machine's configuration includes the content of its work tapes, its current state, and the location of its heads, including the head on the input tape. For convenience, we can assume that the heads locations and current state are written on dedicated worktapes.

functions $B_i\colon [w]\times\{0,1\}\to[w]$ for $i\in[n]$. The ROBP naturally defines a function $B\colon\{0,1\}^n\to\{0,1\}^n$ as follows: Start at $v_0=v_{st}$, and for $i=1,\ldots,n$, read the input symbol x_i and transition to the state $v_i=B_i(v_{i-1},x_i)$. The ROBP accepts x, i.e., $B(x)=1$, if and only if $v_n=v_{ac}$.

In the *adaptive* read-once model, each computation path of the branching program can read the bits of input $r\in\{0,1\}^n$ in a different order, as long as each bit is read exactly once.

Definition 3.9 (AOBP)**.** A (read-once) adaptive order branching program (AOBP) B of width w and length n, is a layered 2-out-regular directed graph with $n+1$ layers, each layer having w vertices, which is also equipped with a labeling function $l\colon V\to[n]$ where V denotes the set of vertices of B, and includes a start and accept vertices v_{st},v_{ac}.

The AOBP defines a function $B\colon\{0,1\}^n\to\{0,1\}$ as follows. Start at $v_0=v_{st}$, and for $i=1,\ldots,n$, transition to the state $v_i=B(v_{i-1},x_{l(v_{i-1})})$, where $B(u,\sigma)$ denotes the σ^{th} neighbor of u in B. The AOBP accepts x, i.e., $B(x)=1$, if and only if $v_n=v_{ac}$. Moreover, we require that for every possible input $x\in\{0,1\}^n$, every bit of x is read at most once over the computation.

When we refer to the *size* of a branching program, we mean the number of vertices of the underlying layered directed graph, namely $(n+1)\cdot w$.

3.3 Pseudorandomness Primitives

We recall several standard definitions here, and mention the explicit constructions in the corresponding technical sections.

Error-correcting codes. We say that an error correcting code $\mathcal{C}\colon\Sigma^k\to\Sigma^n$ has *relative distance* δ if for any distinct codewords $x,y\in\mathcal{C}$, it holds that $\delta(x,y)=\Pr_{i\in[n]}[x_i\neq y_i]\geq\delta$. We say that $\mathcal{C}$ is (ρ,L) list decodable if for any $w\in\Sigma^n$ there are at most L codewords $c\in\mathcal{C}$ that satisfy $\delta(w,c)\leq 1-\rho$; we then refer to ρ as the *agreement* parameter. As is customary, we often use $\mathcal{C}$ to denote $\mathrm{Im}(\mathcal{C})\subseteq\Sigma^n$.

Samplers. We recall the definition of a (strong) sampler:

Definition 3.10 (strong sampler)**.** A function $\mathsf{Samp}\colon\{0,1\}^m\times[t]\to\{0,1\}^n$ is a strong (ε,δ) (oblivious) sampler if for any $H_1,\ldots,H_t\subseteq\{0,1\}^n$ it holds that

$$\Pr_{x\leftarrow\{0,1\}^m}\left[\left|\Pr_{i\leftarrow[t]}[\mathsf{Samp}(x,i)\in H_i]-\mathop{\mathbb{E}}_{i\leftarrow[t]}[\rho(H_i)]\right|\leq\varepsilon\right]\geq 1-\delta,$$

where we denote by $\rho(H_i)=\frac{|H_i|}{2^n}$ the density of a set. The parameter ε is the *accuracy* parameter of the sampler, and δ is its *confidence* parameter.

k-wise independence. We say that a distribution $\mathbf{Z}$ over $\{0,1\}^n$ is k-wise independent with marginal p if for any $I=\{i_1,\ldots,i_k\}\subseteq[n]$ it holds that $\mathbf{Z}|_I=\mathbf{U}^p_{|I|}$, where $\mathbf{U}^p$ is i.i.d Bern(p). It is well-known that one can efficiently sample from a k-wise independent distribution over $\{0,1\}^n$ using $O(k\log n)$ bits, and this holds even for biased marginals:

Claim 3.11 ([Jof74, ABI86])**.** *For any space-constructible $k\colon\mathbb{N}\to\mathbb{N}$ and $p\colon\mathbb{N}\to[0,1]$ such that $p(n)$ can be computed in space $O(\mathrm{loglog}(1/p))$, there is a sequence of k-wise independent distributions $\mathbf{Z}=\{\mathbf{Z}_n\}$ where $\mathbf{Z}_n$ is over $\{0,1\}^n$ and has marginal probability p, such that $\mathbf{Z}$ can be sampled with $\ell(n)=O(k\cdot\log(n/p))$ bits, and the map $(\sigma,i)\to\mathbf{Z}(\sigma)_i$ can be computed in space $O(\log(k)+\mathrm{loglog}(1/p)+\mathrm{loglog}(n))$, where for $\sigma\in\{0,1\}^\ell$, we denote by $\mathbf{Z}(\sigma)$ the σ^{th} element in $\mathbf{Z}$.*

The space complexity analysis can be found, e.g., in [DPT24, Claim 3.17], where it is analyzed for $p = \frac{1}{2}$, but is easily extended to an arbitrary p.[19]

We recall a standard concentration bound for sums of k-wise independent variables.

Theorem 3.12 ([BR94])**.** *Let k be even, let $\mathbf{X}$ be a sum of k-wise independent random variables in $[0,1]$, and let $\mu = \mathbb{E}[\mathbf{X}]$. Then*

$$\Pr[|\mathbf{X} - \mu| > a] \leq \left(\frac{k\mu + k^2}{a^2}\right)^{k/2}.$$

4 New Distinguish-To-Predict Transformations

In this section, we develop our new distinguish-to-predict transformations. In Section 4.1 we present the D2P for Nisan's PRG composed with the "random walks" distinguisher (see Section 2.1.2), in Section 4.2 we present the D2P for the generator of van Melkebeek and Prakriya composed with the "unique shortest paths" distinguisher, and in Section 4.3 we present the D2P for the Forbes-Kelley PRG composed with AOBPs.

4.1 Distinguish-To-Predict for Compositions With the Nisan Generator

We state our results in graph-theoretic terminology, relying on the well-known interpretation of Nisan's PRG as a derandomized graph squaring operation. For completeness, after stating the main D2P result, we explain how to interpret this result using a more classical interpretation wherein Nisan's generator fools ROBPs (see comment after Theorem 4.8).

4.1.1 Graph notation and Nisan's PRG

We now set up how we work with random walks on graphs, and recall the Nisan PRG. For a graph G on n vertices, we let $\mathbf{G} \in [0,1]^{n \times n}$ denote its random walk matrix. Let us first recall how to modify a graph so that it is 2^t-outregular.

Definition 4.1 (canonicalization)**.** For a graph G on n vertices, and $t \in \mathbb{N}$, we let G_t be the 2^t-outregular graph on n vertices where for $\sigma \in \{0, \ldots, 2^t - 1\}, i \in [n]$ and $j \in [n]$, we have $G_t[i, \sigma] = j$ if and only if

$$\sum_{l=1}^{j-1} \mathbf{G}_{i,j} \leq 2^{-t} \cdot \sigma < \sum_{l=1}^{j} \mathbf{G}_{i,j}.$$

Note that G_t can be computed in space $O(t + \log n)$ given access to G and t. We recall a standard bound on the error induced by this edge duplication. Recall that we use the ℓ_∞ matrix norm, which corresponds to the maximum ℓ_1 norm of a row of the matrix.

Claim 4.2. *For every G and t, we have*

$$\|\mathbf{G} - \mathbf{G}_t\| \leq n \cdot 2^{-t}.$$

Next, we define the Nisan PRG, which we do in terms of recursive powering of graphs. First, we recall the hash family we will use:

[19] Specifically, we utilize the fact that the [Jof74] construction gives us k-wise independence over $\mathbb{F}_q$, and choose $q = 2^\ell$ so that $\ell = \lceil \max\{\log n, \log(1/p)\} \rceil$. The corresponding sample space coordinate is 1 if and only if the first $\log(1/p)$ bits of the $\mathbb{F}_q$-symbol are 1, where we handle marginals which are not powers of 2 in a standard manner. The seed length and space complexity extends easily.

Fact 4.3. For every $t \in \mathbb{N}$, there exists a pairwise independent hash family $\mathcal{H}\colon \{0,1\}^t \to \{0,1\}^t$ such that $|\mathcal{H}| = 2^{2t}$, and $h \in \mathcal{H}$ (which we associate with $h \in \{0,1\}^{2t}$) can be evaluated in space $O(\log t)$.

Next, we define the graph obtained from applying a single hash function.

Definition 4.4. Given a 2^t-outregular graph G_t and $h\colon \{0,1\}^t \to \{0,1\}^t$, let $G_{t,h}$ be the graph with adjacency function $G_{t,h}[i,\sigma] = G_t[G_t[i,\sigma], h(\sigma)]$. For a pair of hash functions (h_1, h_2), we let $G_{t,(h_1,h_2)} = (G_{t,h_1})_{h_2}$, and extend to a sequence of hash functions in the natural way.

We recall that we can compute the walk matrices of such graphs space efficiently:

Claim 4.5. *There is a space $O(t + \ell + \log n)$ algorithm that, given G, t, and $h_1, \ldots, h_\ell$, returns $\mathbf{G}_{t,(h_1,\ldots,h_\ell)}$.*

Next, we recall what it means for a hash function to be good for a graph.

Definition 4.6. Let H be 2^t-outregular graph. We say that $h\colon \{0,1\}^t \to \{0,1\}^t$ is ε-good for H if $\|\mathbf{H}_h - \mathbf{H}^2\| \le \varepsilon$.

Finally, we recall the key result that for every graph, most hash functions are good:

Lemma 4.7 ([Nis92, PP23])**.** *For every ε, and 2^t-outregular graph H, we have that, letting h be drawn uniformly from a pairwise-independent hash family on $\{0,1\}^t$,*

$$\Pr[h \text{ is } \varepsilon\text{-good for } H] \ge 1 - (n/\varepsilon)^3/2^t.$$

4.1.2 The deterministic D2P transformation

We can now state our main result as follows:

Theorem 4.8. *For every $n = 2^\ell$, let $t = 50\log(n)$. For every graph G on n vertices, let $T_G\colon (\{0,1\}^{2t})^\ell \to \{0,1\}$ be defined[20] as*

$$T_G(h_1, \ldots, h_\ell) = \bigwedge_{i \in [\ell]} \mathbb{I}\left[h_i \text{ is } n^{-3}\text{-good for } G_{t,(h_1,\ldots,h_{i-1})}\right].$$

Then the following hold:

1. ***Evaluability.*** *The function T_G can be computed in space $O(\log n)$, given G.*
2. ***Usefulness.*** *For every $\vec{h} = (h_1, \ldots, h_\ell)$ such that $T_G(\vec{h}) = 1$, it holds that*
$$\|\mathbf{G}_{t,h_1,\ldots,h_\ell} - \mathbf{G}^n\| \le n^{-2}.$$
3. ***Likeliness.*** *We have $\mathbb{E}[T_G(\mathbf{U}_{\ell \cdot 2t})] \ge 1 - n^{-2}$.*
4. ***D2P.*** *There is a logspace algorithm that, given G, outputs a δ-distinguish to ρ-predict D2P transformation $(\mathsf{PRED}_1, \ldots, \mathsf{PRED}_{b=\mathrm{polylog}(n)})$ for T_G, where $\delta = 1/2$ and $\rho = \Omega(1/\log^2(n))$. Moreover, there is a logspace algorithm that, given G, $i \in [b]$, and x, returns $\mathsf{PRED}_i(x)$.*

[20] Where we interpret the input as a sequence of hash functions per Fact 4.3.

As mentioned above, Theorem 4.8 can be equivalently presented as a D2P for Nisan's PRG composed with (any) ROBP (i.e., rather than with the "random walks" distinguisher). To see this, recall that given an ROBP B of length and width n, we can produce a graph $G = G_B$ on n^2 vertices such that if $T_G(\vec{h}) = 1$ then composing B with $\mathrm{NIS}_{\vec{h}}$ approximately maintains the probability of reaching any vertex v in the last layer of B (i.e., $\Pr_{r\in\{0,1\}^n}[B(r) = v] \approx \Pr_{s\in\{0,1\}^{\ell'}}[B(\mathrm{NIS}_{\vec{h}}(s)) = v]$, where $\ell' = O(\log n)$ is the seed length of $\mathrm{NIS}_{\vec{h}}$ for a fixed $\vec{h}$). Thus, for every distribution $\mathbf{w}$ over sequences of hash functions, if B is a distinguisher for $\mathrm{NIS}_{\mathbf{w}}(\mathbf{u}_{\ell'})$, then T_G is a distinguisher for $\mathbf{w}$, in which case Theorem 4.8 transforms T_G into a list of predictors.

Proof of Theorem 4.8. For Item 1, we test each condition in sequence, where we can compute $\mathbf{G}_{t,h_1,\ldots,h_{i-1}}$ and $\mathbf{G}_{t,h_1,\ldots,h_i}$ in logspace from Claim 4.5. We can then compute the square of the former matrix in logspace, and computing the ℓ_1 distance between two matrices can likewise be done in logspace, so the result follows from Proposition 3.6.

For Item 2, fix an arbitrary G and $\vec{h} = (h_1, \ldots, h_\ell)$ such that $T_G(\vec{h}) = 1$, we have that $\|\mathbf{G}-\mathbf{G}_t\| \le n^{-3}$ by our choice of t and Claim 4.2. Then, by the condition of the test we have that for every i, $\|\mathbf{G}_{t,h_1,\ldots,h_i} - \mathbf{G}^2_{t,h_1,\ldots,h_{i-1}}\| \le n^{-3}$, and the claims follows by induction.

Item 3 follows directly from Lemma 4.7 and the fact that we consider a truly uniform distribution over hash functions, so for an arbitrary prefix $(h_1, \ldots, h_{i-1})$, the probability that the next hash function will be good for $\mathbf{H} = \mathbf{G}_{t,h_1,\ldots,h_{i-1}}$ is at least $1 - n^{-3}$.

Item 4 is the most involved. Here, we appeal to the result of [LPT24] that to construct a D2P transformation, it suffices to construct a PCAPP algorithm. In more detail, by Lemma 3.5, to prove our claim it suffices to construct a logspace PCAPP algorithm with error n^{-2}.

Lemma 4.9. *There is an (n^{-2})-PCAPP algorithm for T_G that, given G, can be evaluated in logspace.*

Thus, to conclude the proof, we just need to prove Lemma 4.9. The algorithm works as follows. Fix an arbitrary input

$$h_< = (h_1, \ldots, h_i, y),$$

where $y \in \{0,1\}^j$ denotes the suffix in the final block. We first test for every $k \le i$ if h_k is n^{-3}-good for $G_{t,(h_1,\ldots,h_{k-1})}$, and note that this test can be done in space $O(t+\log(n)) = O(\log n)$ by Claim 4.5. If this occurs for any k, note that

$$T_G(h_< \circ z) = 0$$

for every suffix z, so we return 0 and estimate the expectation with zero error. Otherwise, we enumerate over $s \in \{0,1\}^{2t-j}$, and compute

$$\rho = \Pr_{s\in\{0,1\}^{2t-j}}\left[h = (y \circ s) \text{ is } n^{-3}\text{-good for } G_{t,(h_1,\ldots,h_i)}.\right]$$

Next, we show that ρ is a good solution for PCAPP, i.e. a good estimate of $\mathbb{E}_z[T_G(h_< \circ z)]$. Note that

$$\left|\rho - \mathop{\mathbb{E}}_z[T_G(h_< \circ z)]\right| \le \max_{s\in\{0,1\}^{2t-j}}\left\{\left|b_s - \mathop{\mathbb{E}}_z[T_G(h_< \circ s \circ z)\right|\right\},$$

where

$$b_s = \mathbb{I}\left[h = (y \circ s) \text{ is } n^{-3}\text{-good for } G_{t,(h_1,\ldots,h_i)}\right].$$

For an arbitrary s, if $b_s = 0$ we have that $T_G(h_< \circ s \circ z) = 0$ for every z so

$$\left|b_s - \mathop{\mathbb{E}}_z[T_G(h_< \circ s \circ z)]\right| = 0.$$

Otherwise if $b_s = 1$, we have

$$\begin{aligned}\mathop{\mathbb{E}}_{z}[T_G(h_< \circ s \circ z)] &\geq \Pr_z[\text{all subsequent hash functions are good}] \\ &\geq 1 - \ell \cdot n^{-3}\end{aligned}$$

so in all cases we have

$$\left| b_s - \mathop{\mathbb{E}}_{z}[T_G(h_< \circ s \circ z)] \right| \leq n^{-2}$$

and the error is bounded as claimed. Finally, enumerating over s and computing $\rho = \mathbb{E}_s[b_s]$ can be done in space $O(t + \log n)$, so we have that the function is logspace computable. □

4.2 Distinguish-To-Predict for Compositions With the van-Melkebeek-Prakriya Generator

Let $G = (V, E)$ be a directed graph over n vertices. The isolation lemma tells us that if we assign edge weights independently and uniformly at random (where each edge gets assigned a weight in $[n^{O(1)}]$), then with high probability, for each $s, t \in V$, there is at most one path of minimum weight from s to t in G. More precisely, a random $W \colon E \to [n^c]$ would be *min-isolating* with probability at least $1 - n^{-(c-4)}$. Li, Pyne, and Tell [LPT24] constructed a deterministic logspace D2P transformation for a slightly stricter requirement. Namely, given some fixed ordering on the edges, $E = \{e_1, \ldots, e_m\}$, they defined

$$T_G(W) = \bigwedge_{i \in [m]} \mathbb{I}\left[W_i \text{ is min-isolating in } G_i\right],$$

where $G_i = (V, E_i = \{e_1, \ldots, e_i\})$ and $W_i = W|_{E_i}$. However, to obtain Theorem 6, investing $\mathrm{poly}(n)$ random bits to generate W will be too costly for us.

4.2.1 The generator of van Melkebeek and Prakriya

In [vMP19], van Melkebeek and Prakriya showed how to generate a min-isolating weight function using only $O(\log^2 n)$ bits, and we now describe their construction.[21] For simplicity (and also since it suffices for Theorem 6), let us assume that our graphs are layered, with number of layers being a power of 2. Concretely, let $V \subseteq [w] \times \{0, 1, \ldots, \ell\}$ (so $n = w(\ell+1)$), $E \subseteq \bigcup_{i \in [\ell]} V_{i-1} \times V_i$ where each $V_i = V \cap ([w] \times \{i\})$, and we assume that $\ell = 2^t$ for some $t \in \mathbb{N}$. The generator of [vMP19] assigns weights iteratively, where in the k^{th} iteration, we assign weights (using the same randomness) to one in every 2^k layers. The weight function will assign weights to vertices rather than edges, but in the case of layered graphs, we can simply reassign the weight of a vertex to each of its outgoing edges.

To formally present the generator, let $\mathcal{H}_{n,M} \subseteq [n] \to [M]$ be a universal family of hash functions for some $M = M(n)$ (think of $M = \mathrm{poly}(n)$).[22] For $k = 1, \ldots, t$, at the k^{th} iteration we assign weights to vertices in $V \setminus \bigcup_{i=0}^{2^{t-k}} V_{i \cdot 2^k}$ as follows.

- Start with $W_0 \equiv 0$.

[21] In fact, they also gave a weight assignment generator that uses $O(\log^{3/2} n)$ bits, but this saving will be immaterial for us.

[22] We want that for every distinct $x, y \in [n]$, and any $\sigma_1, \sigma_2 \in [M]$, it holds that $\Pr_{h \sim \mathcal{H}_{n,M}}[\sigma_1 + h(x) = \sigma_2 + h(y)] \leq 1/M$. Explicit constructions allow us to sample from $\mathcal{H}_{n,M}$ using $O(\log(nM))$ random bits, with each $h \sim \mathcal{H}_{n,M}$ being logspace computable.

- Assume we already constructed W_k, and think about G as consisting of $\ell/2^{k+1}$ consecutive blocks of length 2^{k+1}, where the i^{th} block is the subgraph induced by the vertices in layers $(i-1)2^{k+1}$ through $i \cdot 2^{k+1}$. Consider some i^{th} block $B = B_1 \circ B_2$ of length 2^{k+1}, where each B_i is of length 2^k, sharing a middle layer M in common. By our assumption, W_k assigned weights to all layers apart from B's initial and final layers, and from M.

  The assignment W_{k+1} extends W_k by assigning weights to all such middle layers, namely, the vertices in

$$L_{k+1} = \bigcup_{i \in \{1,3,\ldots,2^{t-k}-1\}} V_{i \cdot 2^k}$$

  as follows. Sample $h_{k+1} \in \mathcal{H}_{n,M}$ uniformly at random, and for every $u \in L_{k+1}$, assign $W_{k+1}(u) = h_{k+1}(u)$.

- The final weight function is given by $W = W_t$.

Thus, we see that sampling W can be done using $s = t \cdot s_h$ bits, where $s_h = O(\log(nM))$ is the number of bits needed to describe a single hash function. Note that $s = O(\log^2 n)$ when $M, \ell = \text{poly}(n)$, and moreover, given $y \in \{0,1\}^s$, we can output the corresponding weight function in $O(\log n)$ space. We denote this weight assignment generator by

$$G_{\mathsf{vMP}} \colon (\{0,1\}^{s_h})^t \to (E \to [M]) \,.$$

Keeping the above notation, the following lemma follows from the analysis in [vMP19].

Lemma 4.10. *For every $k \in [t]$ the following holds. Fix some W_{k-1} that is min-isolating for every subgraph of G induced by the vertices in layers $(i-1)2^{k-1}$ to $i \cdot 2^{k-1}$, where $i \in [2^{t-k-1}]$ (this holds trivially when $k = 1$). Then, with probability at least $1 - \frac{w^4 \ell}{M}$ over the choice of $h_k \in \mathcal{H}_{n,M}$, the weight function W_k is min-isolating for every subgraph of G induced by the vertices in layers $(i-1)2^k$ to $i \cdot 2^k$, where $i \in [2^{t-k}]$.*

As a corollary, choosing $M = n^8$, for every directed layered graph G over n vertices, $G_{\mathsf{vMP}}(U_s)$ is min-isolating for G with probability at least $1 - n^{-2}$.

We set some additional notation towards the next section. Given a layered graph G, and $k \in [t]$, we let $G^{(k)}$ be the collection of induced subgraphs as above, and let $G^{(k)}_{\mathsf{vMP}}$ be the generator that produces the (partial) weight function W_k. Under this terminology, Lemma 4.10 tells us that if $\vec{h} = h_1, \ldots, h_{k-1} \in \{0,1\}^{(k-1)s_h}$ are such that $G^{(k-1)}_{\mathsf{vMP}}(\vec{h})$ is min-isolating for $G^{(k-1)}$, then with probability at least $1 - n^{-3}$ over $h_k \in \{0,1\}^{s_h}$, it holds that $G_{\mathsf{vMP}}(\vec{h}, h_k)$ is min-isolating for $G^{(k)}$ (under the choice $M = n^8$, which we fix here onwards).

4.2.2 A D2P trasformation for randomness-efficient weights generation

Given a layered directed graph G with n vertices and ℓ layers, we let $T_G \colon (\{0,1\}^{s_h})^t \to \{0,1\}$ be defined as

$$T_G(h_1, \ldots, h_t) = \bigwedge_{k \in [t]} T^{(k)}_G(h_1, \ldots, h_k) = \bigwedge_{k \in [t]} \mathbb{I}\left[G^{(k)}_{\mathsf{vMP}}(h_1, \ldots, h_k) \text{ is min-isolating for } G^{(k)} \right].$$

We will construct a D2P transformation via a PCAPP algorithm, similar to what we did in Section 4.1 (and what was done in [LPT24]). And again, we have a *polarization* effect: If the partial assignment has already failed to produce a min-isolation assignment, then T_G is already 0 for every suffix, and otherwise, almost all suffixes will successfully generate a min-isolating assignment.

Theorem 4.11. *For every directed layered graph $G = (V, E)$ with n vertices, and $\ell = 2^t$ layers, let T_G be the above indicator. Then, the following hold.*

- ***Evaluability.*** *Given G, the function $T_G \colon \{0,1\}^{m=O(\log^2 n)} \to \{0,1\}$ can be computed in* $\mathsf{USPACE}[O(\log n)] \cap \mathsf{coUSPACE}[O(\log n)]$.
- ***Usefulness.*** *For every $\vec{h} \in \{0,1\}^s$ such that $T_G(\vec{h}) = 1$, it holds that $G_{\mathsf{vMP}}(\vec{h}) \in (E \to [n^8])$ is min-isolating. Moreover, $\mathbb{E}[T_G(\mathbf{U}_m)] \geq 1 - n^{-2}$.*
- ***D2P.*** *For any $\delta \geq n^{-2}$, there exists a deterministic logspace algorithm $\mathcal{T}$ that, given a layered directed graph G with n vertices, outputs a $(3\delta m)$-distinguish to δ-predict transformation $(P_1, \ldots, P_b)$ for T_G, where $b = O(m^2/\delta)$. Moreover, there exists a* $\mathsf{UL} \cap \mathsf{coUL}$ *machine that on input (G, i, x), computes $P_i(x)$ unambiguously.*

The evaluability property follows Allender and Reinhardt's algorithm [RA00] for testing if a fixed assignment induces unique shortest paths (see also [LPT24, Theorem 6.6]), and the fact that the weights can be computed from the hash functions in logspace. The usefulness property follows from Lemma 4.10. Towards establishing D2P, we start with the following claim.

Claim 4.12. *There exists a nondeterministic TM $\mathcal{M}$, that on input G as above, and $h_1, \ldots, h_i$ for $i \leq t$, runs in space $O(\log n)$ and satisfies the following.*

1. *There exists exactly one computation path on which $\mathcal{M}$ does not output $\perp$,*
2. *On that computation path, $\mathcal{M}$ outputs $T_G^{(i)}(h_1, \ldots, h_i)$, and,*
3. *It holds that*
$$\left| T_G^{(i)}(h_1, \ldots, h_i) - \mathop{\mathbb{E}}_{r \leftarrow \mathbf{U}_{(t-i)s_h}} [T_G(h_1, \ldots, h_i, r)] \right| \leq n^{-2}.$$

Proof. The machine $\mathcal{M}$ iterates over $j = 1, \ldots, i$, and for each j calls the [RA00] algorithm on the graph $G^{(j)}$ and the weight function $G_{\mathsf{vMP}}(h_1, \ldots, h_j)$.[23] If there exists a j for which the [RA00] algorithm returned $\perp$, $\mathcal{M}$ halts and returns $\perp$. If not, $\mathcal{M}$ returns 1 if all j-s returned 1, and otherwise returns 0.

For Items 1 and 2, note that if every $(h_1, \ldots, h_j)$ is good for $G^{(j)}$, there exists exactly one computation path that returns 1 whereas the rest return $\perp$, and otherwise one computation path returns 0 and the rest return $\perp$. Also, the fact that $\mathcal{M}$ runs in logspace is immediate, since we invoke at most n logspace algorithms and the weight functions are logspace computable. For Item 3, if $T_G^{(i)}(h_1, \ldots, h_i) = 0$ then $T_G(h_1, \ldots, h_i, r) = 0$ by definition, for all r. Otherwise, Lemma 4.10 tells us that $T_G(h_1, \ldots, h_i, r) = 1$ with probability at least $1 - (t-i)n^{-3} \geq 1 - n^{-2}$. □

Next, we give a $\mathsf{UL} \cap \mathsf{coUL}$ PCAPP algorithm for T_G. It is almost implied by Claim 4.12 above, but we need to handle the case of prefixes of *every* length.

Lemma 4.13. *There exists a nondeterministic logspace n^{-2}-*PCAPP *algorithm for T_G, such that on input G and a prefix x, there exists exactly one computation path on which the algorithm does not output $\perp$, and that output is the* PCAPP *approximation.*

[23] A subtle technicality is that the [RA00] algorithm also gets two vertices s and t, such that if the weight function does induce unique shortest paths *and* there exists a path from s to t, then there exists exactly one computation path that returns 1. Thus, we feed the algorithm with the endpoints of an edge in $G^{(j)}$. Note that if the weight function does not induce unique shortest paths, exactly one guess sequence will return 0 regardless of s and t.

Proof. We are given G and $x = (h_1, \ldots, h_{i-1}, y)$, where $y \in \{0,1\}^{<s_h}$ corresponds to the bits of x that do not yet describe a hash function. The algorithm runs over all $s \in \{0,1\}^{s_h - |y|}$ and runs $\mathcal{M}$ from Claim 4.12 on G and $(h_1, \ldots, h_{i-1}, h_i = y \circ s)$. If all runs returned a value (either 0 or 1), the PCAPP algorithm returns the average of those values. Otherwise, it returns $\perp$. The fact that the algorithm is unambiguous and runs in logspace follows from Claim 4.12. Next, we observe that

$$\begin{aligned}\left| \mathop{\mathbb{E}}_{s,r \leftarrow \mathbf{U}} [T_G(x \circ s \circ r)] - \mathop{\mathbb{E}}_{s \leftarrow \mathbf{U}} [M(G, x \circ s)] \right| &\leq \max_s \left| \mathop{\mathbb{E}}_{r \leftarrow \mathbf{U}} [T_G(x \circ s \circ r)] - M(G, x \circ s) \right| \\ &\leq n^{-2},\end{aligned}$$

where by $M(\cdot)$ we mean the unique non-$\perp$ value that is guaranteed to exist. Thus, the above algorithm is indeed a PCAPP one. □

Finally, the D2P part of Theorem 4.11 follows readily from Lemma 3.5, recalling that each P_i calls the PCAPP algorithm once, and accepts if and only if the (non-$\perp$) output is above a certain threshold.

4.3 Distinguish-To-Predict for Compositions With the Forbes–Kelley Generator

Finally, we build a version of the Forbes–Kelley generator [FK18] supporting a deterministic D2P transformation. For our application (where the predictor must be incredibly space efficient, whereas the algorithm producing the predictor can use additional space), we actually construct a *white-box Yao transformation* instead: we give an algorithm which, given a distribution that does not fool the distinguisher, finds a suffix such that the Yao predictor with this suffix obtains good advantage. We remark that a very similar approach would give a D2P transform where the candidate predictors are **SC** algorithms.

Theorem 4.14. *For every $\varepsilon > 0$, there is a generator $\mathsf{FK}\colon \{0,1\}^{n^\varepsilon} \to \{0,1\}^n$ with the following properties:*

- ***Fooling.*** *FK fools AOBPs of size n to within error $1/n^2$.*
- ***Computability.*** *The map $(x, j) \to \mathsf{FK}(x)_j$ can be computed in space $O(\varepsilon \cdot \log n)$ with catalytic access to j.*[24]
- ***White-Box Yao Derandomization.*** *There is an algorithm that runs in space $O(n^\varepsilon)$ and time $\mathrm{poly}(n)$ that works as follows. Given input an AOBP $M\colon \{0,1\}^n \to \{0,1\}$ and a distribution $\mathbf{D}$ over $\{0,1\}^{n^\varepsilon}$ such that*

$$|\mathbb{E}[M \circ \mathsf{FK}(\mathbf{D})] - \mathbb{E}[M \circ \mathsf{FK}(\mathbf{U})]| \geq \frac{1}{10},$$

the algorithm outputs one of the following:

1. *A machine $L\colon \{0,1\}^{n^\varepsilon} \to \{0,1\}$ with description size $O(\log n)$ that can be computed in space $\log(n) + O(\varepsilon \log n)$ such that L predicts $\mathbf{D}$ with advantage $1/n^{O(\varepsilon)}$.*

[24] The algorithm is given access to a special read-write tape, initialized to (the binary representation of) j. When the algorithm halts and returns $\mathsf{FK}(x)_j$, the tape must be restored to that initial configuration.

2. $i \in [n^\varepsilon]$, $\sigma \in \{0,1\}^2$, *and* $z \in \{0,1\}^{n^\varepsilon - i}$ *such that*

$$P(x_{<i}) = (M \circ \mathsf{FK}(x_{<i} \circ \sigma_1 \circ z)) \oplus \sigma_2$$

predicts $\mathbf{D}$ *with advantage* $1/n^{O(\varepsilon)}$.

We first define the PRG in terms of n^ε (which we w.l.o.g. assume is a power of two) and then scale ε by an appropriate constant. Let

$$p = n^{-\varepsilon}, \qquad \ell = 10 \log(n) n^\varepsilon$$

and note that we have chosen ℓ so that

$$n \cdot (1 - p/2)^\ell \le n^{-3}.$$

For s to be set later and $\tau = n^{3\varepsilon}$, let

$$\mathsf{FK} \colon (\{0,1\}^s \times \{0,1\}^s)^\ell \times \{0,1\}^\tau \to \{0,1\}^n$$

where we denote the input to the generator as

$$\mathsf{FK}(a_1, b_1, \ldots, a_\ell, b_\ell, v).$$

We define the generator as follows. Let $G \colon \{0,1\}^s \to \{0,1\}^n$ be a $\lceil 12/\varepsilon \rceil$-wise independent generator with marginal $1/2$, and $G' \colon \{0,1\}^s \to \{0,1\}^n$ be a $\lceil 12/\varepsilon \rceil$-wise independent generator with marginal $p = 1 - n^{-\varepsilon}$. We use the generator of Claim 3.11, so we have that $s = O(\log n)$ and the maps $(a_i, j) \to G(a_i)_j$ and $(b_i, j) \to G'(b_i)_j$ can be computed in space $O(\log \log n)$.

Let $A_i = G(a_i)$ and $B_i = G'(b_i)$. Then, for $i \in [\ell]$,

$$\mathsf{FK}_i = A_i \oplus (B_i \wedge \mathsf{FK}_{i+1}).$$

We say that $j \in [n]$ is alive at level i if $(B_1)_j = \cdots = (B_{i-1})_j = 1$. We say B_i eliminates j if j is alive at level i but $(B_i)_j = 0$. Moreover, fixing $B_1, \ldots, B_\ell$, let

$$\mathcal{L} = \{j : j \text{ is alive at level } \ell\}$$

be the set of remaining live variables, and let $\mathcal{L}(j)$ be the index of $j \in [n]$ in this set. Then, define $\mathsf{FK}' = \mathsf{FK}_{\ell+1}(v)$ (recall that $v \in \{0,1\}^\tau$) as follows:

$$(\mathsf{FK}')_j = \begin{cases} v_{\mathcal{L}(j)} & j \in \mathcal{L} \text{ and } \mathcal{L}(j) \le \tau \\ 0 & j \in \mathcal{L} \text{ and } \mathcal{L}(j) > \tau \\ 0 & j \notin \mathcal{L} \end{cases}.$$

Finally, let $\mathsf{FK} = \mathsf{FK}_1$. Note that the final seed length of the generator is

$$s_{\mathsf{FK}}(n) = 2\ell \cdot O(\log n) + \tau \le 2n^{3\varepsilon}.$$

Computability. Recall that we have read-only access to $(a_i, b_i)_{i\in[\ell]}$ and v, and catalytic access to j. First, note that

$$\mathsf{FK}(x)_j = \sigma \oplus \bigoplus_{l<k} (A_l)_j$$

where k is the largest number such that j is alive at level k, and σ is only nonzero if $k = \ell$. We first compute k, which is easy to do with read-only access to (b_i) using 2 counters in $[\ell]$ and the computability guarantee of Claim 3.11. Once we have computed k, we compute $d = \oplus_{l<k}(A_l)_j$ analogously. If $k < \ell$, we then return b, which overall requires space $o(\log n) = O(\varepsilon \log n)$.

Otherwise, we must compute $\mathcal{L}(j)$, the index of j in the set of remaining live variables, and then we let $\sigma = v_{\mathcal{L}(j)}$. We will do so catalytically. We initialize a counter $L = 1$ for the number of live variables encountered so far, and iteratively decrease j by 1, and after each decrease test if $j' < j$ is alive. If so, we increment L and continue. If L ever reaches τ, we cease decrementing and prepare to return 0. Otherwise, once j reaches zero, we have that $L = \mathcal{L}(j)$. In both cases, we then initialize a second counter $L' = 0$ and begin to increment the catalytic space, and for each j' that is live, increment L'. Once $L' = L$, we halt, and it is clear from the description that we successfully restore j. If $L = \tau$, we return 0, and otherwise return $d \oplus v_L = G(x)_j$. As both counters never exceed $\tau = n^{3\varepsilon}$, the workspace usage is as claimed.

Fooling. We recall (a modification of) a lemma in [CLTW23], that states the parameters required to fool AOBPs.

Lemma 4.15 ([CLTW23], Lemma 17)**.** *Let M be a size s AOBP. Let $\mathbf{D}$ be $2k$-wise independent and $\mathbf{T}$ be a $2k$-wise independent distribution with marginal $1-p$. Then*

$$|\mathbb{E}[M(\mathbf{U})] - \mathbb{E}[M(\mathbf{D} \oplus \mathbf{T} \wedge \mathbf{U})]| \le s \cdot p^{k/2}.$$

In [CLTW23], the lemma is only proven for $p = 1/2$. However, inspecting the proof, the only change needed is in Claim 18, where we use the fact that if $\mathbf{T} \sim \{0,1\}^n$ is k-wise independent with marginal p, and $\alpha = \{i_1, \dots, i_k\} \subseteq [n]$, then $\Pr[\forall i \in [k], \mathbf{T}_{\alpha_i} = 0] = p^k$.

We use this lemma to prove that the PRG construction fools AOBPs. First, every variable is still alive after level ℓ with probability at most $(1-p)^\ell \le n^{-4}$, so with probability at least $1 - 1/2n^2$, no variables remain alive by stage ℓ (and hence this level fools the remaining AOBP, which depends on no inputs, with error 0). Furthermore, applying Lemma 4.15 with $s = n, p = p$, and $k = \lceil 6/\varepsilon \rceil$, every preceding level maintains the expectation up to error

$$n^2 \cdot (n^{-\varepsilon})^{6/2\varepsilon} \le \frac{1}{\ell} \cdot \frac{1}{2n^2},$$

so the final error is as claimed.

White-Box Yao derandomization. Next, we construct the white-box derandomization, which is the most involved component, and the reason for the changes compared to the standard Forbes–Kelley instantiation. To do so, we first prove that except with negligible probability, for every sufficiently large set of variables that are alive at level i, a random B_i decreases the number of live variables by at least a factor of $(1 - p/2)$.

Lemma 4.16. *Let $S \subseteq [n]$ be of size at least $n^{3\varepsilon}$. Then with probability at least $1 - 1/n^2$, we have that at least $|S|(p/2)$ variables of the variables in S are unselected in $\mathbf{T} = G'(\mathbf{U})$.*

This follows from applying Theorem 3.12 with $a = (p/2) \cdot |S|$ and $k = \lceil 12/\varepsilon \rceil$.

We will now define a set of distinguishers.

Definition 4.17 (live variables distinguisher)**.** For $i \in [\ell]$, let $L_i \colon (\{0,1\}^s)^i \to \{0,1\}$ be defined as $L_i(b_1, \ldots, b_i) = 0$ if there are more than $n^{3\varepsilon}$ live variables after level $i-1$, *and* dividing these live variables into sequential blocks $P_1, \ldots, P_d$ of size $n^{3\varepsilon}$ (where the final block is of size at most $2n^{3\varepsilon}$), $G'(b_i)$ eliminates fewer than $|P_j| \cdot (p/2)$ variables in some block P_j.

We now prove several properties about these distinguishers. We first prove that solving PCAPP on all but the last indices is trivial.

Claim 4.18 (strong prefix polarization)**.** *For every $i \in [\ell]$ and (possibly empty) prefix $x \in (\{0,1\}^s)^{<i}$, $\mathbb{E}_z[L_i(x \circ z)] \geq 1 - 1/n$.*

Proof. Without loss of generality (as we could consider a worst-case extension of a shorter prefix), consider a prefix $x \in (\{0,1\}^s)^{i-1}$. First, suppose this prefix causes fewer than $n^{3\varepsilon}$ variables to live. In this case, we have $\mathbb{E}_z[L(x \circ z)] = 1$ by definition. Otherwise, the result follows from Lemma 4.16, as a random assignment to the final block will eliminate at least $|P_j| \cdot (p/2)$ variables in each P_j (note that $|P_j| \geq n^{3\varepsilon}$) with probability at least $1 - 1/n^2$, and there are at most n such blocks. □

Next, we show that all these distinguishers can be evaluated very space efficiently.

Claim 4.19 (strong explicitness)**.** *Given $i \in [\ell]$ and $\vec{b} \in (\{0,1\}^s)^i$, $L_i(\vec{b})$ can be evaluated in space $\log(n) + O(\varepsilon \log n)$.*

Proof. We maintain four counters:

1. j, which ranges over $[n]$ and tracks the current active bit.
2. L, which tracks the number of live variables in the current block.
3. K and K_{prev}, which track the number of variables eliminated in the current and previous blocks respectively.

The algorithm increments $j = 1, \ldots, n$ and works as follows. For a fixed j, we determine if j is still live at level $i-1$ (resp. i) and if so increment L (resp. K). This can be performed in space $O(\varepsilon \log n)$ via Claim 3.11. Once L reaches $n^{3\varepsilon}$, we first determine if the previous block failed to eliminate enough variables. In particular, if $K_{prev} < n^{3\varepsilon}(1 - p/2)$ we return 0. Otherwise, set $K_{prev} = K$ and $L = K = 0$ and continue to increment j. Once j reaches n, we reject if and only if the final block fails to eliminate enough variables, i.e. $K_{prev} + K < (n^{3\varepsilon} + L)(1 - p/2)$. □

Next, for a string $\vec{b} = (b_1, \ldots, b_\ell) \in (\{0,1\}^s)^\ell$, we say $\vec{b}$ is good if every level i in which more than $n^{3\varepsilon}$ variables are alive, B_i eliminates at least a $(1 - p/2)$ fraction of variables, and otherwise call a string bad. For convenience later, we say $(\vec{a}, \vec{b})$ is bad if $\vec{b}$ is bad. Note that there is a straightforward space $O(\log n)$ algorithm that tests if $\vec{b}$ is bad.

Claim 4.20. *Let $\vec{b}$ be bad. Then there is an $i \in [\ell]$ such that $L_i(\vec{b}) = 0$.*

Proof. Consider the level i where B_i does not eliminate a sufficient fraction of variables. Next, consider the blocks $P_1, \ldots, P_d$ tested by L_i. By a simple averaging argument, there is some block P_l wherein B_i eliminates an insufficient fraction of variables, and so L_i will reject. □

Theorem 4.21 (PCAPP solver for good restrictions)**.** *There is an $\mathbf{SC}$ algorithm that, given an AOBP M and $\vec{a} = (a_1, \ldots, a_i), \vec{b} = (b_1, \ldots, b_i)$ where $\vec{b}$ is good, returns $\rho \in \mathbb{R}$ so that*

$$\left| \rho - \mathop{\mathbb{E}}_{z} \left[M \circ \mathsf{FK}\left((\vec{a}, \vec{b}) \circ z \right) \right] \right| \leq \frac{1}{n^2}.$$

Proof. Fix an arbitrary such M and $\vec{a}, \vec{b}$. Note that for every variable j that has been eliminated by level l for $l \leq i$, we have

$$x_j = \mathsf{FK}((\vec{a}, \vec{b}) \circ z)_j = (A_1)_j \oplus \cdots \oplus (A_l)_j,$$

i.e., the j^{th} bit of output does not depend on the suffix z. Let $M_{\vec{a},\vec{b}}$ be the AOBP where every such variable j is fixed to x_j, and all other variables are left unfixed. We claim that:

$$\left|\mathbb{E}[M_{\vec{a},\vec{b}}(\mathbf{U})] - \mathop{\mathbb{E}}_{z}[M \circ \mathsf{FK}((\vec{a}, \vec{b}) \circ z)]\right| \leq 1/2n^2. \tag{1}$$

We first show the result assuming Equation (1). Note that we can produce $M_{\vec{a},\vec{b}}$ in logspace given $(\vec{a}, \vec{b})$, and the white-box problem of estimating the expectation of an AOBP is equivalent to estimating the expectation of an ROBP (as the variable order does not matter), and hence can be performed in **SC** by [Nis92].

Proving Equation (1) We now prove the equation. Let L be the number of variables not eliminated by $\vec{b}$. By the fact that $\vec{b}$ is good, we have that

$$L \leq \max\{n^{3\varepsilon}, n(1-p/2)^i\}.$$

We analyze based on the two cases. In both cases, denote the suffix of the generator seed as

$$(\vec{a}', \vec{b}', w) \in (\{0,1\}^s)^{\ell - i} \times (\{0,1\}^s)^{\ell - i} \times \{0,1\}^{n^{3\varepsilon}}$$

and recall that the first section is used for the remaining restrictions and the latter is used for filling in remaining live variables with true randomness.

1. First, suppose that $L \leq n^{3\varepsilon}$. Applying Lemma 4.15 iteratively, we have that

$$\left|\mathbb{E}[M_{\vec{a},\vec{b}}(\mathbf{U}) - \mathop{\mathbb{E}}_{\vec{a}',\vec{b}'}[A_{\vec{a}\circ\vec{a}',\vec{b}\circ\vec{b}'}(\mathbf{U})]\right| \leq \frac{1}{2n^2}.$$

   Furthermore, clearly, further restrictions cannot increase the number of live variables, and hence for every possible $\vec{a}', \vec{b}'$ we have that $A_{\vec{a}\circ\vec{a}',\vec{b}\circ\vec{b}'}$ has at most $n^{3\varepsilon}$ live variables. Thus, averaging over the final component w of the generator is exactly equivalent to supplying random input to $M_{\vec{a}\circ\vec{a}',\vec{b}\circ\vec{b}'}$, so the result follows.

2. Next, suppose that $L > n^{3\varepsilon}$ (and thus $L \leq n(1-p/2)^i$).

   Claim 4.22. *Over a random suffix z, all variables are eliminated with probability at least $1 - 1/2n^2$.*

   Proof. We claim that $i \leq \ell/2$, and then the result follows since $(1-p)^{\ell/2} \leq n^{-4}$, so every variable is eliminated with the claimed probability by a union bound. Assuming otherwise, we have $L < n(1-p/2)^{\ell/2} < 1$, which is a contradiction to the case we are in. □

   Given this, the proof is direct, again using that each of the $\ell - l$ further restrictions preserves the expectation of $M_{\vec{a},\vec{b}}$ up to error $1/(\ell \cdot 2n^2)$, and the final level of the generator will affect 0 variables. □

Finally, we can combine these ingredients and obtain a white-box Yao derandomization that outputs a predictor with advantage $n^{-c\varepsilon}$ for some (universal) constant c. Given an AOBP M and a distribution $\mathbf{D}$ that does not $(1/10)$-fool $M \circ \mathsf{FK}$, we first determine

$$\delta_0 = \Pr_{(\vec{a},\vec{b},v)\leftarrow\mathbf{D}}\left[\vec{b} \text{ is bad}\right].$$

If $\delta_0 > n^{-10\varepsilon}$, we construct a predictor using the tests L_i:

Claim 4.23. *Suppose $\delta_0 > n^{-10\varepsilon}$. Then there is $i \in [\ell]$, $\sigma \in \{0,1\}^2$,and $z \in \{0,1\}^{<s}$, such that the following is an $(n^{-12\varepsilon})$-predictor for $\mathbf{D}$, letting $t = s - |z| - 1$:*

$$f(x_{<t}) = L_i(x_{<t} \circ \sigma_1 \circ z) \oplus \sigma_2.$$

Proof. Since for every $\vec{b} = (b_1, \ldots, b_\ell)$ that is bad contains some index i where b_i is bad, we have by Claim 4.20 and an averaging argument that there is some i where $\mathbb{E}[L_i(\mathbf{D})] < 1 - \delta_0/\ell$, and hence by Claim 4.18 we have that $\mathbf{D}$ does not $(\delta_0/\ell - n^{-1})$-fool L_i. Then, applying Theorem 3.3 with $P = L_i$ and $\delta = (\delta_0/\ell - n^{-1})$, we have that a $\delta/\ell s$-predictor of the form $L_i(x \circ \sigma_1 \circ z) \oplus \sigma_2$ exists, and moreover if the predictor is at bit j,

$$|\mathbb{E}[L_i(\mathbf{D}_{<j} \circ \mathbf{U})] - \mathbb{E}[L_i(\mathbf{D}_{<j+1} \circ \mathbf{U})]| \geq \delta/\ell s.$$

But note that $\delta/\ell s > 1/n$, so by Claim 4.18 this can only occur at indices j in the final block, so the predictor must have that form. □

Thus, from now on we assume $\delta_0 \leq n^{-10\varepsilon}$. Let $\mathbf{D}_1 = \mathbf{D}|\left\{\vec{b} \text{ is good}\right\}$ be the distribution over strings $(\vec{a}, \vec{b}, v)$ in $\mathbf{D}$ such that $\vec{b}$ is good, and note that we can enumerate over $\mathbf{D}_1$ in space $O(\log n)$ given $\mathbf{D}$, as we can determine if an element of $\mathbf{D}$ is in $\mathbf{D}_1$ by testing if $\vec{b}$ is good in space $O(\log n)$. Moreover, an α-predictor for $\mathbf{D}_1$ is an $(\alpha - \delta_0)$-predictor for $\mathbf{D}$ (and that we can produce $\mathbf{D}_1$ from $\mathbf{D}$ in logspace). We will find such a predictor by performing a hybrid argument, then fixing bits. First, note that

$$\begin{aligned}\frac{1}{10} - \delta_0 \leq &\left|\mathbb{E}[M \circ \mathsf{FK}(\mathbf{D}_1)] - \mathop{\mathbb{E}}_{(\vec{a},\vec{b})\leftarrow\mathbf{D}_1, v\leftarrow\mathbf{U}}[M \circ \mathsf{FK}((\vec{a},\vec{b}),v)]\right| \\ &+ \left|\mathop{\mathbb{E}}_{(\vec{a},\vec{b})\leftarrow\mathbf{D}_1, v\leftarrow\mathbf{U}}[M \circ \mathsf{FK}((\vec{a},\vec{b}),v)] - \mathbb{E}[M \circ \mathsf{FK}(\mathbf{U})]\right|.\end{aligned}$$

Furthermore, note that we can estimate all of the quantities in this expression up to error n^{-2} by Theorem 4.21, since we can enumerate over strings in $\mathbf{D}_1$, and for every prefix $(\vec{a}, \vec{b})$ of this string, we are guaranteed that $\vec{b}$ is good. Next, we break intro cases depending on which term is large.

Hybrid Case 1. We first determine if

$$\frac{1}{40} \leq \left|\mathop{\mathbb{E}}_{(\vec{a},\vec{b},v)\leftarrow\mathbf{D}_1}[M \circ \mathsf{FK}((\vec{a},\vec{b}),v)] - \mathop{\mathbb{E}}_{(\vec{a},\vec{b})\leftarrow\mathbf{D}_1, v\leftarrow\mathbf{U}}[M \circ \mathsf{FK}((\vec{a},\vec{b}),v)]\right|$$

By a further hybrid argument (and that we can estimate the quantities to error $n^{-2} \ll 1/\tau$ in SC), we can find $j \in [\tau]$ such that

$$\frac{1}{40\tau} \leq \left|\mathop{\mathbb{E}}_{(\vec{a},\vec{b},v)\leftarrow\mathbf{D}_1, v'\leftarrow\mathbf{U}}[M \circ \mathsf{FK}((\vec{a},\vec{b}), v_{\leq j} \circ v'_{>j})] - \mathop{\mathbb{E}}_{(\vec{a},\vec{b},v)\leftarrow\mathbf{D}_1, v'\leftarrow\mathbf{U}}[M \circ \mathsf{FK}((\vec{a},\vec{b}), v_{<j} \circ v'_{\geq j})]\right|$$

We iteratively fix elements of v' while approximately preserving this gap in expectation, which allows us to obtain a predictor. Since we can enumerate over elements $(\vec{a}, \vec{b}, v) \leftarrow \mathbf{D}_1$ (and can thus create the restricted branching program $M_{(\vec{a},\vec{b}),v_{<j}}$), we can in SC estimate both terms to high accuracy, so there is a straightforward $O(\tau)$-space algorithm that outputs such a good v', and hence we can find a good predictor.

Hybrid Case 2. If this does not occur, we have

$$\frac{1}{40} \leq \left| \mathop{\mathbb{E}}_{(\vec{a},\vec{b})\leftarrow\mathbf{D}_1, v\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a},\vec{b}), v)] - \mathbb{E}[M \circ \mathsf{FK}(\mathbf{U})] \right|.$$

By a further hybrid argument (and that we can estimate the quantities to error $n^{-2} \ll 1/\ell$ in SC) we can find $i \in [\ell]$ such that

$$\frac{1}{40\ell} \leq \left| \mathop{\mathbb{E}}_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow\mathbf{D}_1,(\vec{a}_{>i},\vec{b}_{>i},v)\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a},\vec{b}), v)] - \mathop{\mathbb{E}}_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow\mathbf{D}_1,(\vec{a}_{\geq i},\vec{b}_{\geq i},v)\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a},\vec{b}), v)] \right|$$

We iteratively fix elements of the uniform distribution while approximately maintaining this gap in expectation, which allows us to obtain a predictor. Formally:

Lemma 4.24. *Given $z = (a_{i+1}, b_{i+1}, \ldots, a_l, b_l)$ such that*

$$\Pr_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow\mathbf{D}_1} [(\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \text{ is bad}] \leq \rho, \qquad \Pr_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow\mathbf{D}_1,(a_i,b_i)\leftarrow\mathbf{U}} [(\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \text{ is bad}] \leq \rho,$$

and

$$\alpha \leq \left| \mathop{\mathbb{E}}_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow\mathbf{D}_1,(\vec{a}_{>l},\vec{b}_{>l},v)\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a},\vec{b}), v)] - \mathop{\mathbb{E}}_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow\mathbf{D}_1,(\vec{a}_{>l},\vec{b}_{>l},v,a_i,b_i)\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a},\vec{b}), v)] \right|$$

we can find in SC a pair (a_{l+1}, b_{l+1}) such that fixing the block to this value results in an expectation gap of at least $\alpha - \rho - 4/n$, and moreover

$$\Pr_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow\mathbf{D}_1} [(\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ (a_{l+1}, b_{l+1}) \text{ is bad}] \leq \rho + 1/n,$$

and

$$\Pr_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow\mathbf{D}_1,(a_i,b_i)\leftarrow\mathbf{U}} [(\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ (a_{l+1}, b_{l+1}) \text{ is bad}] \leq \rho + 1/n.$$

We apply Lemma 4.24 as follows. Initially, we apply it with an empty suffix, and take $\rho = 1/n^2$. We apply the lemma in ℓ stages, where ρ is eventually bounded by ℓ/n, and at each application we lose in α the current value of ρ. Thus, in space $\widetilde{O}(n^{\varepsilon})$ and time $\mathrm{poly}(n)$ we can find a string $z = (a_{i+1}, b_{i+1}, \ldots, a_\ell, b_\ell)$ such that

$$\begin{aligned} \frac{1}{40\ell} - \ell^2/n \leq \Bigg| & \mathop{\mathbb{E}}_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow\mathbf{D}_1, v\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ v)] \\ & - \mathop{\mathbb{E}}_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow\mathbf{D}_1,(a_i,b_i)\leftarrow\mathbf{U}, v\leftarrow\mathbf{U}} [M \circ \mathsf{FK}((\vec{a}_{<i},\vec{b}_{<i}) \circ (a_i, b_i) \circ z \circ v)] \Bigg| \end{aligned}$$

Next, we attempt to fix bits of v to maintain this gap. For every good prefix, solving PCAPP on v is clearly in **SC** (as it is equivalent to estimating the expectation of an ROBP), and so we find in space $O(n^{3\varepsilon})$ and polynomial time a string v such that

$$\frac{1}{40\ell} - 2\ell^2/n \leq \left| \mathop{\mathbb{E}}_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow \mathbf{D}_1} [M \circ \mathsf{FK}((\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ v)] \right.$$
$$\left. - \mathop{\mathbb{E}}_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow \mathbf{D}_1,(a_i,b_i)\leftarrow \mathbf{U}} [M \circ \mathsf{FK}((\vec{a}_{<i},\vec{b}_{<i}) \circ (a_i,b_i) \circ z \circ v)] \right|$$

Finally, applying Theorem 3.3[25], we have that enumerating over the poly(n) possible assignments to (suffixes of) the bits of (a_i, b_i) and $\sigma \in \{0,1\}^2$, at least one such assignment results in a predictor for $\mathbf{D}_1$ with advantage $\frac{1}{60s\ell}$, and hence a predictor for $\mathbf{D}$ with advantage $\frac{1}{60s\ell} - \delta_0$.

Proof of Lemma 4.24. Let

$$\beta_{a,b} = \mathop{\mathbb{E}}_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow \mathbf{D}_1,(\vec{a}_{>l+1},\vec{b}_{>l+1},v)\leftarrow \mathbf{U}} [M \circ \mathsf{FK}((\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ (a,b) \circ (\vec{a}_{>l+1},\vec{b}_{>l+1}), v)]$$

and

$$\beta'_{a,b} = \mathop{\mathbb{E}}_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow \mathbf{D}_1,(\vec{a}_{>l},\vec{b}_{>l},v,a_i,b_i)\leftarrow \mathbf{U}} [M \circ \mathsf{FK}((\vec{a}_{<i},\vec{b}_{<i}) \circ (a_i,b_i) \circ z \circ (a,b) \circ (\vec{a}_{>l+1},\vec{b}_{>l+1}), v)].$$

Also, let

$$\rho_{a,b} = \Pr_{(\vec{a}_{\leq i},\vec{b}_{\leq i})\leftarrow \mathbf{D}_1} [(\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ (a_l,b_l) \text{ is bad}],$$

and

$$\rho'_{a,b} = \Pr_{(\vec{a}_{<i},\vec{b}_{<i})\leftarrow \mathbf{D}_1,(a_i,b_i)\leftarrow \mathbf{U}} [(\vec{a}_{\leq i},\vec{b}_{\leq i}) \circ z \circ (a_l,b_l) \text{ is bad}].$$

Note that given (a,b), we can estimate $\beta_{a,b}$ and $\beta'_{a,b}$ up to error $n^{-2}+\rho_{a,b}$ and $n^{-2}+\rho'_{a,b}$ respectively by Theorem 4.21, and we can compute $\rho_{a,b}$ and $\rho'_{a,b}$ in logspace. The algorithm enumerates over candidates (a,b), verifies if $\rho_{a,b} \leq \rho + 1/n$ and $\rho'_{a,b} \leq \rho + 1/n$, and if so, estimates $|\beta_{a,b} - \beta'_{a,b}|$ and returns the (a,b) which maximizes this quantity.

To show that this algorithm returns a good enough (a,b) (in fact, that it returns anything at all), we first establish that almost all (a,b)'s give rise to ρ's within the desired range. To do so, we use an auxiliary lemma.

Claim 4.25. *For every distribution* $\mathbf{H}$ *supported on good* $(\vec{a},\vec{b})$*, at least a* $1 - 1/n$*-fraction of pairs* (a,b) *satisfy*

$$\Pr_{(\vec{a},\vec{b})\leftarrow \mathbf{H}} [(\vec{a},\vec{b}) \circ (a,b) \text{ is bad}] \leq 1/n.$$

Proof. Let $\tau_{a,b} = \mathbb{E}_{(\vec{a},\vec{b})\leftarrow \mathbf{H}} \left[\mathbb{I} \left[(\vec{a},\vec{b}) \circ (a,b) \text{ is bad} \right] \right]$. We have $\mathbb{E}_{a,b}[\tau_{a,b}] \leq 1/n^2$, as for every fixed string $(\vec{a},\vec{b})$, it follows from Lemma 4.16 that only an n^{-2} fraction of the (a,b)'s are bad. Then, the result follows by Markov. □

[25] Technically, we apply the proof, using that an α-gap in expectation on s bits implies an α/s predictor on one such bit.

Applying the claim with $\mathbf{H} = (\mathbf{D}_1)_{\leq i} \circ z$ and $\mathbf{H} = (\mathbf{D}_1)_{<i} \circ \mathbf{U} \circ z$ (where we remove the ρ fraction of bad elements in both cases), we have that at least a $1-2/n$ fraction of the (a,b)'s satisfy $\rho_{a,b} \leq \rho + 1/n$ and $\rho'_{a,b} \leq \rho + 1/n$. This gaurantees that the algorithm will always find a valid pair (a,b). Finally, note that

$$\alpha \leq \left| \mathop{\mathbb{E}}_{a,b} \beta_{a,b} - \mathop{\mathbb{E}}_{a,b} \beta'_{a,b} \right| \leq \mathop{\mathbb{E}}_{a,b} \left[|\beta_{a,b} - \beta_{a,b}| \right] \leq 1$$

Applying reverse Markov with $X = \mathbb{E}_{a,b} \left[|\beta_{a,b} - \beta_{a,b}| \right]$, we have that

$$\Pr[X > \alpha - 4/n] \geq \frac{\alpha - (\alpha - 4/n)}{1 - 4/n} \geq \frac{5}{n}$$

so with probability at least $5/n$ over (a,b), $|\beta_{a,b} - \beta'_{a,b}| \geq \alpha - 4/n$. Thus there exists (a,b) that simultaneously satisfies this and has $\max\{\rho_{a,b}, \rho'_{a,b}\} \leq \rho + 1/n$. Since we will be able to estimate both quantities to within error $\rho + 1/n$, our returned pair will have the required gap. □

5 A Generator with Uniform Near-Deterministic Logspace Reconstruction

The main result we prove in this section is a version of the Shaltiel-Umans [SU05] generator that is computable in logspace, and whose reconstruction procedure is a uniform, logspace, near-deterministic algorithm. Very recently, Chen *et al.* [CLO+23] showed a version of the generator in which the reconstruction procedure is a uniform algorithm (rather than a non-uniform circuit). We improve their work by showing a procedure with lower randomness complexity and space complexity.

Theorem 5.1 (a somewhere-PRG with uniform near-deterministic logspace reconstruction)**.** *Let $M \colon \mathbb{N} \to \mathbb{N}$ be a logspace-computable function such that $M(N) \leq N^{\varepsilon_{\mathsf{SU}}}$, where $\varepsilon_{\mathsf{SU}} > 0$ is a universal constant. Then, there exist a pair of algorithms SU and RSU that for every $f \in \{0,1\}^N$ satisfy the following.*

1. *When SU is given input 1^N and oracle access to f it runs in space $O(\log N)$ and prints a collection $L_1, ..., L_\ell$ where each L_i is a list of $\mathrm{poly}(N)$ strings of length $M = M(N)$, where $\ell = O(\log(N)/\log(M))$.*
2. *For each $i \in [\ell]$, let $j_i \leq M$, and let $P_i \colon \{0,1\}^{j_i} \to \{0,1\}$ be a $(1/M^2)$-next-bit-predictor for the uniform distribution on L_i. Then, when RSU gets input 1^N and oracle access to f and to $P_1, ..., P_\ell$, it runs in space $O(\log N)$, uses $O(\log N)$ random coins, and prints a (deterministic) oracle circuit $C \colon \{0,1\}^{\log(N)} \to \{0,1\}$ of size $\mathrm{poly}(M)$ such that with probability at least $1/\mathrm{poly}(M)$ over the coin tosses we have $C^{P_1,...,P_\ell}(x) = f(x)$ for all $x \in [N]$.*

In Section 5.1 we present the arithmetic setting for the generator and reconstruction, as well as a few preliminary technical lemmas. In Section 5.2 we present the generator itself. Then, in Section 5.3 and Section 5.4 we present two stand-alone parts of the reconstruction procedure, and in Section 5.5 we present the full reconstruction procedure.

5.1 Arithmetic Setup

Throughout our argument, we will denote the input by $x \in \{0,1\}^n$ (rather than $f \in \{0,1\}^N$) and the output length by m instead of M. We also assume without loss of generality that $m \geq \log(n)$ (otherwise, the generator can trivially output all m-bit strings).

5.1.1 Arithmetic setting

For input length $n \in \mathbb{N}$ and output length $m \leq n$, and for a constant $\varepsilon = \varepsilon_{\mathsf{SU}} > 0$:

- **(Field.)** Let $q = \Theta(m \cdot \log(n))^c$ be a prime power, where $c > 1$ is a sufficiently large universal constant. We consider $\mathbb{F}_q$ as an extension of a subfield $\mathbb{F}_{q_0}$ of size $q_0 = \Theta(m \cdot \log n)$; note that the extension degree is a constant $\Delta = \Theta(c)$.
- **(Degree.)** Let $d = m^{\varepsilon}$.
- **(Number of variables.)** Let $v = O_{\varepsilon}(\log(n)/\log(m))$ such that $v \geq (1/\varepsilon) \cdot \log(n/\log(q))/\log(d)$.
- **(Prediction advantage.)** Let $\rho = 1/2m^2$.

Given $x \in \{0,1\}^n$, treat it as a list of $\lfloor n/\log(q) \rfloor$ coefficients specifying a polynomial $\hat{x} \colon \mathbb{F}_q^v \to \mathbb{F}_q$ of degree d. By our lower bound on v we have $\binom{d+v}{d} \geq \lfloor n/\log(q) \rfloor$, and therefore all the coefficients specified by x are useful towards defining $\hat{x}$; in particular different x's give rise to different polynomials $\hat{x}$.[26]

Fact 5.2. There is an algorithm that gets input n, m, q, Δ satisfying the constraints above, runs in space $O(\log n)$, and outputs a representation of $\mathbb{F}_q$.

Proof. Let $q = p^r$ for a prime p. The algorithm enumerates over degree-$(r-1)$ polynomials $\mathbb{F}_p \to \mathbb{F}_p$, each of which is represented by $r \cdot \log(p) = \log(q) < O(\log n)$ bits, and tests each polynomial u for irreducibility. The latter test is also done by brute-force, enumerating over all polynomials $\mathbb{F}_p \to \mathbb{F}_p$ of degree at most $r-2$ and checking if there is one that divides u. □

Fact 5.3. There is an algorithm that gets as input a representation of $\mathbb{F}_q$ and an integer $v \in \mathbb{N}$, runs in space $O(v \cdot \log(q))$, and prints a monic irreducible polynomial $u^\star \in \mathbb{F}_q[x]$ of degree $v-1$.

Proof. The algorithm works by brute-force, analogously to the proof of Fact 5.2. Each polynomial $\mathbb{F}_q \to \mathbb{F}_q$ of degree $v-1$ is represented by $O(v \cdot \log(q))$ bits. □

Due to Fact 5.2 and 5.3, from now on we will assume that all of our space-bounded algorithms have access to a fixed representation of $\mathbb{F}_{q^v}$, in the form of the irreducible polynomial $u^\star \in \mathbb{F}_q[x]$ produced by the algorithm above.

5.1.2 A generator matrix in logspace

We will need an algorithm that prints powers of a generator matrix for $\mathbb{F}_q^v$ in space $O(v \cdot \log(q))$. We first define this notion, and show that several basic operations in $\mathbb{F}_{q^v}$ and in $\mathbb{F}_q^v$ can be done in small space. Then, we construct the algorithm for printing powers of a generator matrix.

Definition 5.4. We say that $A \in \mathbb{F}_q^{v \times v}$ is a generator matrix for $\mathbb{F}_q^v$ if $\left\{A^i \cdot \vec{s}\right\}_{i \in [q^v - 1]} = \mathbb{F}_q^v \setminus \left\{\vec{0}\right\}$ for any non-zero $\vec{s} \in \mathbb{F}_q^v$.

Claim 5.5. *There is an algorithm that gets input $A \in \mathbb{F}_q^{v \times v}$ and $i \in [q^v - 1]$, runs in space $O(\log(i) \cdot \log(q))$, and prints A^i.*

[26] When x is too short to specify all the coefficients of a polynomial of degree d, we consider the polynomial $\hat{x}$ obtained by padding x with zeroes to the appropriate length.

Proof. Consider the binary tree of depth $\lceil \log(i) \rceil$ with i leaves labeled by A and $2^{\lceil \log(i) \rceil} - i$ leaves labeled by the identity matrix, and each node labeled by the multiplication of the labels of its children. Computing each entry of the label of each node can be done in space $O(\log(v) + \log(q)) = O(\log q)$ with query access to the labels of its children. The algorithm prints each entry of the top node, and simulates the query access of each node by space-efficient composition; the space complexity is thus $O(\log(i) \cdot \log(q))$. □

Claim 5.6. *There is an algorithm that gets input $\omega \in \mathbb{F}_{q^v}$, runs in space $O(\log(v) \cdot \log(q))$, and prints the matrix $T_\omega \in \mathbb{F}_q^{v \times v}$ that represents multiplication by ω in $\mathbb{F}_q^v$.*[27]

Proof. Let $C_{u^\star} \in \mathbb{F}_q^{v \times v}$ be the companion matrix of the irreducible $u^\star$ from Fact 5.3, and recall that $C_{u^\star} = T_x$ where $x \in \mathbb{F}_q[x]/(u^\star)$ is the identity polynomial. Also recall that $C_{u^\star}$ has a very simple structure,[28] and in particular there is an algorithm that (given $u^\star$) prints $C_{u^\star}$ in space $O(\log(v) + \log(q))$.

Let $\omega = \sum_{i=0}^{v-1} \omega_i x^i$. Then, $T_\omega = \sum_{i=0}^{v-1} \omega_i C_{u^\star}^i$. Using Claim 5.5, we can print each $C_{u^\star}^i$ in space $O(\log(v) \cdot \log(q))$, and hence we can also print T_ω in such space. □

Proposition 5.7. *There is a generator matrix A for $\mathbb{F}_q^v$ and an algorithm A' such that A' gets input $i \in [q^v - 1]$, runs in space $O(\log(n))$, and prints A^i.*

Proof. The algorithm first finds a primitive element $\omega \in \mathbb{F}_{q^v}$, by brute-force. That is, it enumerates over elements of $\mathbb{F}_{q^v}$, and for each element ω' it raises it to the powers $i = 2, 3, ..., q^v - 1$ and checks whether any intermediate result is 1. This can readily be done in space $O(v \log q)$.

Now, let ω be the first primitive element encountered, and recall that $A = T_\omega$ is a generator matrix for $\mathbb{F}_q^v$. The algorithm raises ω to the power i, and then uses Claim 5.6 to compute $T_{\omega^i} = T_\omega^i = A^i$. The proposition follows, noting that $v \log q = O(\log n)$. □

5.1.3 A standard list-decodable code

Our construction will use a logspace-computable list decodable code. We do not need particularly tight parameters, and the classical construction of Sudan, Trevisan, and Vadhan [STV01] suffices for us. (We do not even rely on the locality of the decoder in their construction.)

Theorem 5.8 (a list-decodable code; see [STV01])**.** *There is a universal constant $c_{\mathsf{STV}} > 1$ and algorithm $\mathsf{Enc}_{\mathsf{STV}}$ that maps $x \in \{0,1\}^{\log(q)}$ to $\mathsf{Enc}_{\mathsf{STV}}(x) \in \{0,1\}^{\ell_q = \mathrm{poly}(\log(q), 1/\rho)}$ such that the mapping yields a $(\frac{1}{2} - \rho, \bar{\rho} = (1/\rho)^{c_{\mathsf{STV}}})$-list-decodable code, $\mathsf{Enc}_{\mathsf{STV}}$ runs in space $O(\log q)$, and the list-decoder $\mathsf{Dec}_{\mathsf{STV}}$ runs in time $\mathrm{poly}(\log(q), 1/\rho)$.*

5.2 The Generator

On an input $x \in \{0,1\}^n$, G first encodes x as a polynomial $\hat{x} \colon \mathbb{F}_q^v \to \mathbb{F}_q$ of (total) degree d.

The lists $L_0, ..., L_{v-1}$. We first define "q-ary lists" whose elements are vectors in $\mathbb{F}_q^m$, and then define the final output lists L_i whose elements are strings in $\{0,1\}^m$. For every $i = 0, ..., v-1$, the

[27] That is, consider the $\mathbb{F}_q$-basis $\{1, x, x^2, ..., x^{v-1}\}$ for $\mathbb{F}_q^v$, and the corresponding bijection $\xi \colon \mathbb{F}_{q^v} \to \mathbb{F}_q^v$ (i.e., ξ maps a polynomial $\sum_{i \in \{0,...,v-1\}} a_i x^i$ to $(a_0, ..., a_{v-1})$). Then, for every $\omega, \nu \in \mathbb{F}_{q^v}$ we have that $T_\omega \cdot \xi(\nu) = \xi(\omega \cdot \nu)$.

[28] Specifically, the coefficients of $u^\star$ appear in its rightmost column, and otherwise all of the entries in the matrix are zero except for one subdiagonal whose entries are one.

i^{th} q-ary list $L_i^{(q)} \subseteq \mathbb{F}_q^m$ that $G(x)$ outputs is defined as follows. For every $\vec{s} \in \mathbb{F}_q^v$, the generators includes in $L_i^{(q)}$ the m-element string

$$L_i^{(q)}(\vec{s}) = \hat{x}(A^{1\cdot q^i} \cdot \vec{s}) \circ \hat{x}(A^{2\cdot q^i} \cdot \vec{s}) \circ ... \circ \hat{x}(A^{m\cdot q^i} \cdot \vec{s}), \tag{5.1}$$

where A is the generator matrix given by Proposition 5.7.

Then, for every $\vec{s} \in \mathbb{F}_q^v$ and $j \in [\ell_q]$, the corresponding m-bit string in L_i is

$$L_i(\vec{s}, j) = \mathsf{Enc}_{\mathsf{STV}}\left(L_i^{(q)}(\vec{s})_1\right)_j \circ \mathsf{Enc}_{\mathsf{STV}}\left(L_i^{(q)}(\vec{s})_2\right)_j \circ ... \circ \mathsf{Enc}_{\mathsf{STV}}\left(L_i^{(q)}(\vec{s})_m\right)_j .$$

The list L_v. In addition, the generator outputs the list $L_v \subseteq \{0,1\}^m$ defined as follows. Let $\mathsf{pow}\colon \{0,1\}^{v\cdot\log(q)} \to \{0,1\}^{v\cdot\log(q)}$ be the function that parses its input $i \in \{0,1\}^{v\cdot\log(q)}$ as an integer $i \in \{0,...,q^v-1\}$ and outputs $\mathsf{pow}(i) = A^i \cdot \vec{1}$ (in binary representation). Then, for every $i \in \{0,1\}^{v\cdot\log(q)}$ and $z \in \{0,1\}^{v\cdot\log(q)}$ the generator outputs

$$L_v(i,z) = \left\langle \mathsf{pow}^{(m-1)}(i), z \right\rangle \circ \left\langle \mathsf{pow}^{(m-2)}(i), z \right\rangle \circ ... \circ \langle \mathsf{pow}(i), z\rangle \circ \langle i, z\rangle ,$$

where $\mathsf{pow}^{(j)}$ is the j-wise repeated composition of pow.

Complexity. Note that there are $v+1 = O(\log(n)/\log(m))$ lists, and each list contains at most $q^{2v}\cdot \mathrm{poly}(m) = \mathrm{poly}(n)$ strings of m field elements.

Also note that the generator is computable in space $O(\log n)$. To see this, for each L_i with $i < v$, and for each fixed $\vec{s}$, observe that computing each output element reduces in space $O(\log n)$ to computing $A^i \cdot \vec{s}$, which can be done in space $O(\log n)$ using Proposition 5.7. For L_v, given any fixed $(z,i) \in \{0,1\}^{v\cdot\log(q)} \times \{0,1\}^{v\cdot\log(q)}$, the bottleneck is computing $\mathsf{pow}^{(j)}(i)$. To do so, observe that the output of pow is of length $v\cdot\log(q)$; hence, we can iteratively compute $\mathsf{pow}^{(j)}(i)$ by storing the output of each iteration and computing pow again.

5.3 The Reconstruction Procedure for L_v

Let $P_v\colon \{0,1\}^{i_v} \to \{0,1\}$ be the $(1/m^2)$-next-bit-predictor for L_v, where $i_v < m$.

Proposition 5.9 (efficiently printing a circuit that computes discrete log)**.** *There is an algorithm R_{dl} that on input 1^n runs in space $O(\log n)$, uses $O(\log n)$ random coins, and with probability at least $1/\mathrm{poly}(m)$ outputs an oracle circuit C_{dl} of size $\mathrm{poly}(m)$ such that $C_{\mathsf{dl}}^{P_v}(A^i \cdot \vec{1}) = i$ for all $i \in \{0,1\}^{v\cdot\log(q)}$.*

Proof. The algorithm R_{dl} will be the combination of three algorithms R_1, R_2, R_3 that print three corresponding circuits C_1, C_2, C_3. We first describe the three algorithm, and then explain how to combine them to get a single algorithm and a single circuit.

Consider the oracle circuit C_1 that gets $y = A^i \cdot \vec{1} = \mathsf{pow}(i) \in \{0,1\}^{v\cdot\log(q)}$ and tries to find i. The circuit gets an additional input $z \in \{0,1\}^{v\cdot\log(q)}$, computes

$$w_{y,z} = \left\langle \mathsf{pow}^{(i_v-1)}(y), z \right\rangle \circ \left\langle \mathsf{pow}^{(i_v-2)}(y), z \right\rangle \circ ... \circ \langle \mathsf{pow}(y), z\rangle \circ \langle y, z\rangle ,$$

and outputs $P_v(w_{y,z})$. The distribution obtained by uniformly choosing $i \in \{0,1\}^{v\cdot\log(q)}$ and setting $y = A^i\cdot\vec{1}$ is identical to the distribution obtained by uniformly choosing $y_0 \in \{0,1\}^{v\cdot\log(q)}$ and setting $i = \mathsf{pow}^{(m-1-i_v)}(y_0)$ and $y = A^i \cdot \vec{1}$. With probability at least $1/2 + 1/m^2$ over (y_0, i, y) chosen from

the latter distribution and over z, the predictor satisfies $P_v(w_{y,z}) = \mathsf{pow}^{(m-1-i_v)}(y_0) = i$. Since the distributions are identical, we have that

$$\Pr_{i,z}[C_1^{P_v}(A^i \cdot \vec{1}, z) = \langle i, z \rangle] \geq 1/2 + \varepsilon_1, \tag{5.2}$$

where $\varepsilon_1 = 1/m^2$. Observe that C_1 is of size $\mathrm{poly}(m, \log(n)) = \mathrm{poly}(m)$ and that it can be printed by a machine R_1 running in space $O(\log n)$. For simplicity, we will denote $C_1 = C_1^{P_v}$.

The next procedure will compute the mapping $A^i \cdot \vec{1} \mapsto i$ correctly on $\varepsilon_2 = \mathrm{poly}(\varepsilon_1)$ fraction of the i's. We will use a space-efficient and randomness-efficient version of the Goldreich-Levin [GL89] decoding algorithm, given by Doron, Pyne, and Tell [DPT24] following Pyne, Raz, and Zhan [PRZ23]:

Theorem 5.10 (efficient Goldreich-Levin decoding; see [DPT24, Theorem 5.16])**.** *Let $k = v \cdot \log(q)$. There is an algorithm $\mathsf{Dec}_{\mathsf{GL}}$ that gets input $\varepsilon_1, \delta > 0$ and a random seed s_{GL} of length $O(k + \log(1/\delta))$, runs in space $O(\log(k/\varepsilon_1) + \mathrm{loglog}(1/\delta))$, and outputs a list of $L_{\mathsf{GL}} = O((k/\varepsilon_1^2) \cdot \log(1/\delta))$ oracle circuits $C_{s_{\mathsf{GL}},1}, ..., C_{s_{\mathsf{GL}},L_{\mathsf{GL}}}$ satisfying the following.*

- *Each $C_{s_{\mathsf{GL}},i}$ is an oracle TC^0 circuit of size $\mathrm{poly}(k/\varepsilon_1)$ with one majority gate that makes non-adaptive oracle queries.*
- *For every $i \in \{0,1\}^{v \cdot \log(q)}$ and every $C_1^{(i)} \colon \{0,1\}^{v \cdot \log(q)} \to \{0,1\}$ satisfying $\Pr_z[C_1^{(i)}(z) = \langle i, z \rangle] \geq 1/2 + \varepsilon_1/2$, we have that*

$$\Pr_{s_{\mathsf{GL}}}\left[\exists j \in [L_{\mathsf{GL}}] : \forall u \in [k],\ C_{s_{\mathsf{GL}},j}^{C_1^{(i)}}(u) = i_u\right] \geq 1 - \delta.$$

Consider the algorithm R_2 that gets input $A^i \cdot \vec{1}$, draws a random $s_{\mathsf{GL}} \in \{0,1\}^{O(k+\log(1/\delta))}$ and $j \in [L_{\mathsf{GL}}]$, executes $\mathsf{Dec}_{\mathsf{GL}}$ with values $\varepsilon_1/2$ and $\delta = 1/2$, and prints an oracle circuit C_2 that gets input $A^i \cdot \vec{v}$ and outputs the truth-table of $C_{s_{\mathsf{GL}},j}^{C_1^{(i)}}$, where $C_1^{(i)}(z) = C_1(A^i \cdot \vec{1}, z)$. By Eq. (5.2), with probability at least $\varepsilon_1/2$ over $i \in \{0,1\}^{v \cdot \log(q)}$ it holds that $\Pr_z[C_1(A^i \cdot \vec{1}, z) = \langle i, z \rangle] \geq 1/2 + \varepsilon_1/2$, and for each such i, by Theorem 5.10, with probability at least $(1-\delta)/L_{\mathsf{GL}}$ over s_{GL}, j the truth-table of $C_{s_{\mathsf{GL}},j}^{C_1^{(i)}}$ is i. Hence, $\Pr_{i,s_{\mathsf{GL}},j}[C_2^{C_1}(A^i \cdot \vec{1}) = i] \geq \frac{\varepsilon_1 \cdot (1-\delta)}{2L_{\mathsf{GL}}} = \Omega(\varepsilon_1/(m^4 \cdot \log(n)))$. It follows that with probability at least $\Omega(\varepsilon_1/(m^4 \cdot \log(n)))$ over the random choices s_{GL}, j of R_2, we have that $\Pr_i[C_2^{C_1}(A^i \cdot \vec{1}) = i] \geq \varepsilon_2 = \Omega(\varepsilon_1/(m^4 \cdot \log(n)))$. Note that C_2 is of size $\mathrm{poly}(m, \log(n)) = \mathrm{poly}(m)$, and that R_2 can print it in space $O(\log n)$ (relying on the fact that $\mathsf{Dec}_{\mathsf{GL}}$ runs in space $O(\log(v \cdot \log(q) \cdot m^2)) = O(\log n)$). Also, the number of random coins that R_2 uses is $O(k + \log(1/\delta) + \log(L_{\mathsf{GL}})) = O(v \cdot \log(q) + \log(m)) = O(\log n)$.

The next procedure will compute $A^i \cdot \vec{1} \mapsto i$ correctly on all inputs i, using the random self-reducibility of discrete log. This procedure is a space-efficient and randomness-efficient adaptation of [CLO+23, Lemma 4.6].

For any fixed $j \in [q^v - 1]$, consider the following oracle circuit $C_{3,j}$ (looking ahead, for some choices of j, the $C_{3,j}$'s will be sub-circuits in C_3). Given input $A^i \cdot \vec{1}$ and oracle access to $C_2^{C_1}$, the circuit $C_{3,j}$ computes $\vec{v} = A^j \cdot (A^i \cdot \vec{1})$, and then computes $b = C_2^{C_1}(\vec{v})$. It checks whether $A^b \cdot \vec{1} = \vec{v}$, and rejects otherwise. Now it knows that $A^b \cdot \vec{1} = A^j \cdot A^i \cdot \vec{1}$, and hence $A^{-j} \cdot A^b \cdot \vec{1} = A^i \cdot \vec{1}$, and the circuit outputs $\begin{cases} b - j & b > j \\ q^v - (b - j) & o.w. \end{cases}$. (Recall that A is a generator matrix, and hence A is invertible.) Note that $C_{3,j}$ is of size $\mathrm{polylog}(n)$ and can be printed in space $O(\log n)$. (We rely on Proposition 5.7 to hard-wire a description of A into $C_{3,j}$.)

To construct R_3 and C_3 we will need the following space-efficient sampler:

Theorem 5.11 (see, e.g. [DPT24, Theorem 3.12]). *For every $\varepsilon, \delta \colon \mathbb{N} \to [0,1]$ computable in space $O(\log(1/\varepsilon\delta))$, there is an algorithm* Samp *that for every $n \in \mathbb{N}$ computes a strong (ε,δ)-sampler with sample size* $\mathrm{poly}(\log(1/\delta), \varepsilon)$ *and randomness $\bar{n} = n + O(\log(1/\varepsilon\delta))$, using space $O(\bar{n})$.*

The machine R_3 uses Theorem 5.11 with output length $v \cdot \log(q)$, accuracy $\varepsilon_2/2 = 1/\mathrm{poly}(m, \log(n))$, and confidence $1/n^2$, to draw a random sample. Note that the number of random coins for Samp is $O(v \cdot \log(q) + \log(n)) = O(\log n)$, and that its sample size is $\mathrm{poly}(m, \log(n))$. Then, R_3 prints an oracle circuit C_3 that gets input $A^i \cdot \vec{1}$, computes $C_{3,j}^{C_2^{C_1}}(A^i \cdot \vec{1})$ for every output $j \in \{0,1\}^{v \cdot \log(q)}$ in the sample of Samp, and if one of the $C_{3,j}$'s printed i then C_3 prints that i.[29] By a union-bound, with probability at least $1 - 1/n$, it holds that C_3 computes $A^i \cdot \vec{1} \mapsto i$ for all $i \in \{0,1\}^{v \cdot \log(q)}$. Note that R_3 is computable in space $O(\log n)$, and that C_3 is of size $\mathrm{poly}(m, \log(n))$.

The final algorithm R_{dl} combines R_1, R_2, R_3 in a straightforward way to output an oracle circuit C_{dl} that implements $C_3^{C_2^{C_1}}$ (the oracle queries that C_{dl} makes are intended to be answered by P_v, since C_1 requires oracle access to P_v). The space complexity of R_{dl} is $O(\log n)$, it uses $O(\log n)$ random coins, and conditioned on R_2 being successful (which happens with probability at least $1/\mathrm{poly}(m)$), with high probability it outputs C_{dl} of size $\mathrm{poly}(m, \log(n)) = \mathrm{poly}(m)$ that, when given oracle access to P_v, correctly computes $A^i \cdot \vec{1} \mapsto i$ for all $i \in [q^v - 1]$. □

5.4 The Reconstruction Procedure for $L_0^{(q)}, \ldots, L_{v-1}^{(q)}$

In this section we will construct a procedure that gets oracle access to predictors for the q-ary lists $L_0^{(1)}, \ldots, L_{v-1}^{(q)}$ (in a sense that will be defined in Definition 5.16) and computes the function $y \mapsto \hat{x}(A^y \cdot \vec{1})$. The main result statement appears in Proposition 5.20.

Notation. For $i \in \{0, \ldots, q_0 - 1\}$, let w_i be the $(i+1)^{\text{th}}$ element in $\mathbb{F}_{q_0}$ in lexicographical order. For consistency, throughout the section we will use the following notation:

- i ranges in $\{0, \ldots, v\}$.
- j ranges in $[m-1]$.
- k ranges in $[q^v - 1]$.
- t is an element in $\mathbb{F}_q$, or an index in a set $[r]$.

For brevity, we will also use the following notation for elements of $\mathbb{F}_q$. Recall that $\mathbb{F}_q \equiv \mathbb{F}_{q_0}^{\Delta}$, and we will frequently parse each $t \in \mathbb{F}_q$ as a sequence consisting of one element $w_i \in \mathbb{F}_{q_0}$ and $\Delta - 1$ elements $u \in \mathbb{F}_{q_0}^{\Delta - 1}$. Thus, we will frequently denote this by $t = (w_i, u) \in \mathbb{F}_q$.

5.4.1 Pseudorandom primitives

We now present a few pseudorandomness primitives that we will need for the reconstruction of the q-ary lists $L_0^{(q)}, \ldots, L_{v-1}^{(q)}$. Specifically, we will need the randomness-efficient curve sampler by Guo [Guo13], and a space-efficient averaging sampler. We first define curve samplers, state Guo's result, and then prove that Guo's sampler is computable in logspace. Then, we state the averaging sampler that our proof will use.

[29] Observe that for every j, the circuit $C_{3,j}$ never errs (i.e., it either aborts or outputs the correct answer), so if several $C_{3,j}$'s print answers, the answers are identical.

Throughout this section and the next, we will frequently refer to distributions over subsets S' of a set S as (ε, δ)-samplers. (Recall that the standard terminology refers to samplers as functions whose output is a subset $S' \subseteq S$, i.e., the set of sampled points.) By this terminology, we mean that for every $T \subseteq S$, with probability $1 - \delta$ over the choice of S' we have $\Pr_{s \in S'}[s \in T] \in |T|/|S| \pm \varepsilon$.

We will also frequently identify curves $C \colon \mathbb{F}_q \to \mathbb{F}_q$ with their image $C = \{C(t)\}_{t \in \mathbb{F}_q}$, and it will be clear from context which of the two interpretation of "a curve" we are referring to (i.e., a function or its image). For a matrix $A \in \mathbb{F}_q^{v \times v}$, the notation $A \cdot C$ means $\{A \cdot C(t)\}_{t \in \mathbb{F}_q}$.

Definition 5.12 (curve sampler)**.** Let $\mathsf{Samp} \colon \{0,1\}^{\bar{n}} \times \mathbb{F}_q \to \mathbb{F}_q^v$ be an (ε, δ)-sampler. We say that Samp is a degree-t curve sampler if for every fixed $z \in \{0,1\}^{\bar{n}}$, the function $\mathsf{Samp}_z(i) = \mathsf{Samp}(z, i)$ is a curve $\mathbb{F}_q \to \mathbb{F}_q^v$ of degree at most t.

Theorem 5.13 (Guo's curve sampler [Guo13])**.** *Let $\varepsilon, \delta \colon \mathbb{N} \to [0,1]$ and $q \colon \mathbb{N} \to \mathbb{N}$ be space-computable such that $q(v)$ is a prime power satisfying $q(v) \geq (v \cdot \log(1/\delta(v))/\varepsilon(v))^{\Theta(1)}$. Then, there is an algorithm that for every $v \in \mathbb{N}$ computes a strong degree-t curve (ε, δ)-sampler*

$$\mathsf{Samp} \colon \{0,1\}^{\bar{n}} \times \mathbb{F}_q \to \mathbb{F}_q^v$$

with $t = (m \cdot \log_q(1/\delta))^{O(1)}$ and $\bar{n} = O(v \cdot \log(q) + \log(1/\delta))$, using space $O(v \cdot \log(q) + \mathrm{loglog}(1/\delta)))$.

Proof. We prove that Guo's construction is computable in space $O(v \cdot \log(q) + O(\mathrm{loglog}(1/\delta)))$, and that it yields a strong sampler. Let us first bound the space-complexity. The construction is the composition of an outer sampler and an inner one, and we first analyze them separately and then analyze the composition.

Claim 5.13.1. The outer sampler in Guo's construction $\mathsf{Out} \colon \mathbb{F}_q^{O(v + \log_q(1/\delta))} \times \mathbb{F}_q^{\log(v)+1} \to \mathbb{F}_q^v$ is computable in space $O(v \cdot \log(q) + \mathrm{loglog}(1/\delta))$.

Proof. The outer sampler first transforms its source into a block source, using the condenser of [GUV09]. Given $(x, y) \in \mathbb{F}_q^{\bar{n}_{\mathsf{out}}} \times \mathbb{F}_q$ and a parameter v_i, where $\bar{n}_{\mathsf{out}} = O(v + \log_q(1/\delta))$, the condenser outputs

$$\mathsf{Cond}_{v_i}(x, y) = \big(y, f_x(y), f_x(\zeta \cdot y), ..., f_x(\zeta^{v_i - 2} \cdot y)\big) \in \mathbb{F}_q^{v_i} \tag{5.3}$$

where $\zeta \in \mathbb{F}_q$ is a primitive element and $f_x(z) = \sum_{i=0}^{\bar{n}_{\mathsf{out}}-1} x_i \cdot z^i$. We can find a primitive element $\zeta \in \mathbb{F}_q$ in space $O(\log q)$ (by brute-force), raise it to the power $\leq v_i$ in space $O(\log(v_i) + \log(q))$, and compute $(x, z) \mapsto f_x(z)$ in space $O(\log(\bar{n}_{\mathsf{out}}) + \log(q))$. The transformation of the source into a block source is

$$\mathsf{Blk}(x, y_1, ..., y_s) = (\mathsf{Cond}_{v_1}(x, y_1), ..., \mathsf{Cond}_{v_s}(x, y_s)) \in \mathbb{F}_q^{4(v-1)},$$

where $s = \log(v)$, $v_i = 4v \cdot 2^{-i}$, $\sum_{i \in [s]} v_i = 4(v-1)$, and Blk is computable in space $O(\log(v) + \log(q) + \mathrm{loglog}(1/\delta))$ because Cond is computable in that space.

Now, given a block source $((a_1, b_1), ..., (a_s, b_s))$ where $(a_i, b_i) \in \mathbb{F}_q^{v_i/2} \times \mathbb{F}_q^{v_i/2}$ for all i, and seed $y'_s \in \mathbb{F}_q$, the outer extractor works as follows. For each $i = s, ..., 1$, it prints the first $v_i/2 - 1$ elements of $a_i \cdot y'_i + b_i$ and defines y_{i-1} to be the last element of $a_i \cdot y'_i + b_i$. This is computable in space $O(v \cdot \log(q))$ since the linear function in each block (i.e., $a_i \cdot y'_i + b_i$) is computable in such space, and the algorithm only needs to store a single element in $\mathbb{F}_q$ when moving from one block to the next. By composing this algorithm with Blk, we get an (ε, δ)-sampler

$$\mathsf{Out} \colon \mathbb{F}_q^{O(v + \log_q(1/\delta))} \times \mathbb{F}_q^{\log(v)+1} \to \mathbb{F}_q^v$$

computable in space $O(v \cdot \log(q) + \mathrm{loglog}(1/\delta))$, where the output length is truncated to v (i.e., we rely on the fact that $\sum_{i \in [s]} v_i/2 - 1 = 2v - 2 - \log(v) > v$). □

Claim 5.13.2. The inner sampler $\mathsf{In}\colon \mathbb{F}_q^{\mathrm{polylog}(v)+O(\log_q(1/\delta))} \times \mathbb{F}_q \to \mathbb{F}_q^{\log(v)+1}$ is computable in space $O(\log(q)\cdot\mathrm{polylog}(v))$.

Proof. Let $\ell = \log(v)+1$. The sampler In is defined recursively, with $s' = \log(\ell) = O(\mathrm{loglog}(v))$ levels of recursion. For level i, we fix parameters $d_i = \ell/2^i$ and $t_i = \begin{cases} 16^i/4 & i < s' \\ 16^i/4 + 5\cdot\log_q(1/\delta) & i = s' \end{cases}$, and define an algorithm

$$\mathsf{In}_i\colon \mathbb{F}_q^{4t_i\cdot d_i} \times \mathbb{F}_q^{d_i} \to \mathbb{F}_q^{\ell}.$$

At the first level In_0 just outputs its seed. At level $i \geq 1$, the algorithm In_i gets input $(x_{i,1}, x_{i,2}) \in \mathbb{F}_q^{3t_i\cdot d_i} \times \mathbb{F}_q^{t_i\cdot d_i}$ and a seed $s_i \in \mathbb{F}_q^{d_i}$, computes $(z_{i,1}, z_{i,2}, z_{i,3}) = \mathsf{Curve}_i(x_{i,1}, s_i) \in \mathbb{F}_q^{3d_i}$, and outputs $\mathsf{In}_{i-1}\left(\mathsf{Cond}_i(x_{i,2}, z_{i,1}), (z_{i,2}, z_{i,3})\right)$, where the algorithms are defined as follows.

- The algorithm $\mathsf{Curve}_i(x_{i,1}, s_i)\colon \mathbb{F}_q^{3t_i\cdot d_i} \times \mathbb{F}_q^{d_i} \to \mathbb{F}_q^{3d_i}$ parses its input $x_{i,1}$ as t_i triplets of elements $(c_{0,1}, c_{0,2}, c_{0,3}), \ldots, (c_{t_i-1,1}, c_{t_i-1,2}, c_{t_i-1,3}) \in \mathbb{F}_{q^{d_i}}^3$, parses its seed s_i as an element $y \in \mathbb{F}_{q^{d_i}}$, and computes $\left(\sum_{j=0}^{t_i-1} c_{j,1}\cdot y^j, \sum_{j=0}^{t_i-1} c_{j,2}\cdot y^j, \sum_{j=0}^{t_i-1} c_{j,3}\cdot y^j\right)$. It then parses the latter triplet as $3d_i$ elements in $\mathbb{F}_q$ and outputs these elements.

  Note that Curve_i is computable in space $O(d_i\cdot\log(q)+\log(t_i)) \leq O(\log(v)\cdot\log(q)+\mathrm{polylog}(v))$, and that its output is of length $O(d_i\cdot\log(q)) \leq O(\log(v)\cdot\log(q)))$.

- The algorithm $\mathsf{Cond}_i\colon \mathbb{F}_q^{d_i\cdot t_i} \times \mathbb{F}_q^{d_i} \to \mathbb{F}_q^{2d_i\cdot t_{i-1}}$ parses its input $x_{1,2}$ as t_i elements in $\mathbb{F}_{q^{d_i}}$ and its seed as an element $y \in \mathbb{F}_{q^{d_i}}$, and outputs $2d_i\cdot t_{i-1}$ elements defined as in Eq. (5.3).

  This algorithm works over the field of size $q' = q^{d_i}$, and is computable in space $O(\log(q') + \log(d_i\cdot t_{i-1})) \leq O(\log(v)\cdot\log(q))$. Its output length is at most $t_{i-1}\cdot d_i\cdot\log(q) = \mathrm{polylog}(v)$.

Thus, the computation of $\mathsf{In} = \mathsf{In}_{s'}$ amount to computing two strings of total length at most $\log(q)\cdot\mathrm{polylog}(v)$, and then passing them on to level $s'-1$ (as the input and seed to $\mathsf{In}_{s'-1}$); indeed, at each level, the algorithm maps its input to two strings, and gives these strings as input to the level below. Since the strings at each level are of length at most $\log(q)\cdot\mathrm{polylog}(v)$, and the computation at each level can be done in space $O(\log(v)\cdot\log(q)+\mathrm{polylog}(v))$, the inner sampler is computable in space $O(\log(q)\cdot\mathrm{polylog}(v))$. □

The final sampler uses the inner sampler $\mathsf{In}\colon \mathbb{F}_q^{\bar{n}_{\mathsf{in}}} \times \mathbb{F}_q \to \mathbb{F}_q^{\log(v)+1}$ to sample from the outputs of Out, where $\bar{n}_{\mathsf{in}} = \mathrm{polylog}(v) + O(\log_q(1/\delta))$; that is, given $(x, x') \in \mathbb{F}_q^{\bar{n}_{\mathsf{out}}} \times \mathbb{F}_q^{\bar{n}_{\mathsf{in}}}$ and $y \in \mathbb{F}_q$,

$$\mathsf{Samp}((x, x'), y) = \mathsf{Out}(x, \mathsf{In}(x', y)).$$

The bound on the space complexity follows by combining Claims 5.13.1 and 5.13.2.

Strongness. Having proved that Guo's curve sampler is computable in small space, let us now prove that the construction yields a strong sampler. We only give here a proof sketch, and the full proof involves fully articulating standard techniques from extractor theory.

In [Guo13], the outer and inner samplers are in fact analyzed using the terminology of randomness extractors, and indeed, strong samplers are equivalent to strong extractors [Zuc97]. Following [RSW06, Theorem 8.2], we know that when we compose an outer and an inner extractor, for the final extractor to be strong, it suffices for the outer one to be strong (with a very minor loss in parameters, that essentially "sacrifices" the entropy in the seed).

The outer extractor Out employs the block-source extraction framework, after a block-source conversion step. One can verify that if the block-source conversion step is strong (namely, that Blk is close to a block-source even conditioned on a typical fixing of $y_1, \ldots, y_s$)), and the extractor used in the block-source extraction procedure is strong, then the entire process yields a strong extractor. To argue that the block-source conversion step is strong, one can use the fact that Cond is a strong condenser (the latter fact is immediate, since it outputs the seed y). For the block-source extraction step, Out uses the "line extractor" that maps $((a,b),y)$ to $(a_1 y + b_1, \ldots, a_{v_i} y + b_{v_i})$. The fact that it is strong readily follows from its analysis as a sampler (see, e.g., [Guo13, Lemma 2.3]). □

Having established Theorem 5.13, the following corollary is immediate.

Corollary 5.14. *For any $\varepsilon = \mathrm{poly}(\rho)$ and $\delta = q^{-O(v)}$, there is a probabilistic algorithm that generates a curve $C\colon \mathbb{F}_q \to \mathbb{F}_q^v$, using $O(\log n)$ random coins and in space $O(\log n)$, such that $d^{\mathsf{crv}} \triangleq \deg(C) = (m \cdot \log(n))^{O(1)}$ and the resulting distribution over curves is a strong (ε, δ)-sampler.*

An averaging sampler. Recall that $\mathbb{F}_q \equiv \mathbb{F}_{q_0}^{\Delta}$. We will also need to sample a set of points $R \subseteq \mathbb{F}_{q_0}^{\Delta - 1}$ space-efficiently and randomness-efficiently, which we do using the sampler from Theorem 5.11.

Corollary 5.15. *For any $\delta = q^{-O(v)}$, there is a probabilistic algorithm that generates a set $R \subseteq \mathbb{F}_{q_0}^{\Delta-1}$ of size $r = \mathrm{poly}(v, \log(q))$, using $O(\log n)$ random coins and in space $O(\log n)$, such that the distribution over R's is an (ε, δ)-sampler, for $\varepsilon = \frac{1}{(m \cdot \log(n))^2}$.*

Proof. We use Theorem 5.11 with output length $\Delta \cdot \log(q_0) < \log(q)$ and $\varepsilon = 1/(m \cdot \log(n))^2$ and $\delta = q^{-O(v)}$. The sample size is $\mathrm{poly}(1/\varepsilon, \log(1/\delta)) = \mathrm{poly}(m, \log(n))$, the required randomness is $\log(q) + O(\log(1/\varepsilon\delta)) = O(\log n)$, and the space complexity is linear in the randomness. □

5.4.2 Learning a single curve

We first show an algorithm analogous to "Learn Next Curve" in [SU05]. Intuitively, the algorithm uses a predictor and a sequence of "known" points on previous curves to predict points on the next curve, and then uses a small number of "known" points on the next curve in order to error-correct its predictions. The crucial part for us is the efficiency of this algorithm (i.e., it is a space-bounded machine that outputs a small circuit), and distilling the exact properties that this algorithm needs from the distribution over the relevant curves in order to work.

Definition 5.16 (good predictors)**.** We say that $P^{(i)}\colon \mathbb{F}_q^{m-1} \to \mathbb{F}_q^{\bar{\rho}}$ is a ρ-good predictor for $L_i^{(q)}$ if $\Pr_{\vec{z} \in L_i^{(q)}}[P^{(i)}(\vec{z}_{1,\ldots,m-1}) \ni \vec{z}_m] > \rho$.

Recall that $\bar{\rho}$ was defined as $\bar{\rho} = (1/\rho)^{c_{\mathsf{STV}}}$ in Theorem 5.8, and indeed we use the same parameter when considering predictors in Definition 5.16.

For simplicity of presentation, we will assume throughout this section that all predictors predict the m^{th} element; that is, for each $i \in \{0, \ldots, v-1\}$, the predictor $P^{(i)}\colon \mathbb{F}_q^{j_i} \to \mathbb{F}_q^{\bar{\rho}}$ for $L_i^{(q)}$ has $j_i = m - 1$. This assumption does not meaningfully affect the argument (in fact, it is a "worst-case" scenario for the reconstruction) and we make it only to reduce notational clutter.

Lemma 5.17 (derandomized learning of a single curve)**.** *There is a machine $\mathsf{LrnNext}$ that gets input 1^n and $i \in \{0, \ldots, v-1\}$ and $R \subseteq \mathbb{F}_{q_0}^{\Delta-1}$ of size $r = |R|$, runs in space $O(\log n)$, and prints a circuit $C_{\mathsf{LrnNext},i}$ of size $\mathrm{poly}(q, 1/\rho)$ satisfying the following.*

1. ***Input.*** *Points* $\left\{(a_t^{(m-1)},...,a_t^{(1)}) \in \mathbb{F}_q^{m-1}\right\}_{t\in\mathbb{F}_q}$, *evaluations* $\{b_t \in \mathbb{F}_q\}_{t\in[r]}$, *and* $i^* \in \{0,...,v\}$.

2. ***Output.*** *A set* $\{o_t \in \mathbb{F}_q\}_{t\in\mathbb{F}_q}$.

3. ***Functionality.*** *Consider a distribution over curves* $C\colon \mathbb{F}_q \to \mathbb{F}_q^v$ *of degree* $D = d^{\mathsf{crv}} \cdot q_0 \cdot r$ *and an independent distribution over sets* $R \subseteq \mathbb{F}_{q_0}^{\Delta-1}$ *of size* r *such that the distribution over* C *is a* $(\rho/4, q^{-20v})$*-sampler and the distribution over* R *is a* $(1/2, q^{-20v})$*-sampler. Then, for every* $(\rho/2)$*-good predictor* $P^{(i)}$*, with probability at least* $1 - q^{-10v}$ *over* C, R *it holds that:*

   When $a_t^{(j)} = \hat{x}(A^{-j\cdot q^i} \cdot C(t))$ *for all* $t \in \mathbb{F}_q$ *and* $j \in [m-1]$*, and* $b_t = \hat{x}(C(w_{i^*}, R_t))$ *for all* $t \in [r]$*, and* $C_{\mathsf{LrnNext},i}$ *is given as oracle* $P^{(i)}$*, its output satisfies* $o_t = \hat{x}(C(t))$ *for all* $t \in \mathbb{F}_q$.

Proof. Let us first describe $C_{\mathsf{LrnNext},i}$, and then prove the required properties. The circuit will use Sudan's [Sud97] list-decoding algorithm for the Reed-Solomon code:

Theorem 5.18 (list-decoding of the RS code [Sud97])**.** *Given* p *distinct pairs* $\{(x_a, y_a) \in \mathbb{F}_q \times \mathbb{F}_q\}_{a\in[p]}$*, there are at most* $2/\mu$ *degree-*d' *polynomials* g *such that* $g(x_a) = y_a$ *for at least a* μ*-fraction of the pairs, as long as* $\mu > \sqrt{2d'/p}$*. Furthermore, a list of all such polynomials can be computed in time* $\mathrm{poly}(p, \log(q))$.

For each $t \in \mathbb{F}_q$, the circuit queries its oracle $P_j\colon \mathbb{F}_q^{m-1} \to \mathbb{F}_q^{\bar{\rho}}$ on the point $(a_t^{(m-1)},...,a_t^{(1)}) \in \mathbb{F}_q^{m-1}$, which yields a set $S_t \subseteq \mathbb{F}_q$ of size $|S_t| = \bar{\rho}$. It then uses Theorem 5.18 with the set $S = \cup_{t\in\mathbb{F}_q} S_t$ and with parameter values $p = \bar{\rho}\cdot q$ and $\mu = (\rho/4)\cdot\bar{\rho}$ and $d' = d \cdot D$ to obtain a list of $8/(\rho\cdot\bar{\rho})$ polynomials in time $\mathrm{poly}(q, \rho^{-1})$. (Note that $\mu > \sqrt{2d'/p}$, since $q^{1-1/\Delta} > 32d \cdot d^{\mathsf{crv}} \cdot r \cdot \mathrm{poly}(1/\rho)$, relying on a sufficiently large choice of constant $c > 1$ in Section 5.1.1.) If the list contains a unique polynomial $p\colon \mathbb{F}_q \to \mathbb{F}_q$ such that $p(w_{i^*}, R_t) = b_t$ for all $t \in [r]$, output $\{p(t)\}_{t\in\mathbb{F}_q}$; otherwise, fail.

Observe that there is a uniform machine that gets a first set of inputs $1^n, i, R$ and a second set of inputs $\left\{(a_t^{(m-1)},...,a_t^{(1)})\right\}, \{b_t\}, i^*$, and computes the value of $C_{\mathsf{LrnNext},i}$ (i.e., of the circuit corresponding to the first set of inputs) on the second set of inputs in time $t_{\mathsf{LrnNext}} = \mathrm{poly}(q, \rho^{-1})$. Hence, this functionality is also computable by a $O(\log(t_{t_{\mathsf{LrnNext}}}))$-space-uniform circuit of size $\mathrm{poly}(t_{\mathsf{LrnNext}})$ (i.e., by a standard simulation of machines by highly uniform circuits of quadratic size). The machine $\mathsf{LrnNext}$ gets input $1^n, i, R$, and prints the latter circuit in space $O(\log(t_{t_{\mathsf{LrnNext}}})) \leq O(\log n)$ while hard-wiring the values of i and of R to the appropriate input gates.

Analysis. We want to prove the claim about the functionality of $C_{\mathsf{LrnNext},i}$. We first argue that:

Claim 5.18.1. With probability $1 - q^{-20v}$ over the choice of C, it holds that

$$\Pr_{t\in\mathbb{F}_q}[\hat{x}(C(t)) \in S_t] \geq \rho/4.$$

Proof. Consider a uniformly chosen $\vec{z} \in L_i^{(q)}$. By the guarantee on $P^{(i)}$ we know that

$$\Pr[P^{(i)}(\vec{z}_{1,...,m-1}) \ni \vec{z}_m] \geq \rho/2.$$

However, we also know that $\vec{z}$ is distributed identically to

$$\hat{x}(A^{-(m-1)\cdot q^i}(\vec{y})) \circ ... \circ \hat{x}(A^{-q^i}\cdot\vec{y}) \circ \hat{x}(\vec{y})$$

for a uniformly chosen $\vec{y} \in \mathbb{F}_q^v$. (This is because A^{q^i} is invertible, and relying on the definition of $L_i^{(q)}(\vec{s})$ in Eq. (5.1) and on the fact that $\vec{z}$ is obtained via a uniform choice of $\vec{s} \in \mathbb{F}_q^v$.)

Hence, when choosing a uniformly random $\vec{y} \in \mathbb{F}_q^v$, with probability at least $\rho/2$ we have that

$$P^{(i)}(\hat{x}(A^{-(m-1)\cdot q^i}(\vec{y})), ..., \hat{x}(A^{-q^i} \cdot \vec{y})) \ni \hat{x}(\vec{y}). \tag{5.4}$$

Since the distribution over C is an $(\varepsilon = \mathrm{poly}(\rho), \delta = q^{-20v})$-sampler, with probability at least $1-\delta$, the fraction of points $\vec{y} = C(t)$ on the curve such that Eq. (5.4) holds is at least $\rho/2 - \varepsilon = \rho/4$. □

By Claim 5.18.1, with probability $1 - q^{-20v}$ over the choice of C there are at least $\mu = (\rho/4) \cdot \bar{\rho}$ pairs (t, u) in $S = \cup_{t \in \mathbb{F}} \{t\} \times S_t$ such that $u = p_C(t)$, where $p_C(t) = \hat{x}(C(t))$. Also note that p_C is of degree $d' = d \cdot D$. Hence, for every C satisfying the above, the list of polynomials that the algorithm from Theorem 5.18 outputs contains p_C.

Now, condition on such a C, and consider any $p \neq p_C$ of degree $\deg(p) = d \cdot D$. Note that $d \cdot D / q_0^{\Delta-1} < 1/2$ (relying on a sufficiently large choice of $c > 1$ in the definition of q). Hence, by the Schwartz-Zippel lemma, the fraction of roots of $p - p_C$ in any set $S \subseteq \mathbb{F}_q$ of size $q_0^{\Delta-1}$ is less than $1/2$. In particular,

$$\Pr_{u \in \mathbb{F}_{q_0}^{\Delta-1}} [p(w_{i^*}, u) \neq p_C(w_{i^*}, u)] > 1/2.$$

Since the distribution over $R \subseteq \mathbb{F}_{q_0}^{\Delta-1}$ is a $(1/2, q^{-20v})$-sampler, with probability $1 - q^{-20v}$ there is $t \in [r]$ such that $p(w_i, R_t) \neq p_C(w_i, R_t)$. By a union-bound over all p's that the algorithm of Theorem 5.18 outputs, with probability at least $1 - q^{-20v} \cdot \mathrm{poly}(1/\rho) \geq 1 - q^{-10v}$ over R the circuit $C_{\mathsf{LrnNext},i}$ outputs the unique p_C.[30] □

5.4.3 Learning interleaved curves

The next algorithm will construct two interleaved curves, using Corollary 5.14 and Corollary 5.15, such that one can use the $C_{\mathsf{LrnNext},i}$'s from Lemma 5.17 repeatedly to learn $q^v - 1$ "shifts" of each of these curves by A. At each step, the previous learned curve will intersect the next curve we want to learn at sufficiently many locations for the needed error-correction in Lemma 5.17. Our construction of the interleaved curves is different than that in previous works (e.g., in [SU05, CLO+23]), since we need a randomness-efficient algorithm.

Lemma 5.19 (derandomized interleaved learning)**.** *There is a randomized algorithm that gets input 1^n, uses $O(\log n)$ random coins and $O(\log n)$ space, and outputs two curves $C_1, C_2 \colon \mathbb{F}_q \to \mathbb{F}_q^v$ and a set $R \subseteq \mathbb{F}_{q_0}$ such that for every collection of $(\rho/2)$-good predictors $P^{(0)}, ..., P^{(v)}$, with probability $1 - q^{-5v}$ the following holds. For every $i \in \{0, ..., v-1\}$ and $k \in [q^v - 1]$, when we give* LrnNext *input i and R, it prints $C_{\mathsf{LrnNext},i}$ such that:*

1. **Learning $C^{\mathsf{Nxt},1}(t) \triangleq A^{k+q^i} \cdot C_1(t)$ from $C^{\mathsf{Prv},1}(t) \triangleq A^k \cdot C_2(t)$.** *When we give $C_{\mathsf{LrnNext},i}$ the points $\left\{a_t^{(j)} = \hat{x}(A^{-j\cdot q^i} \cdot C^{\mathsf{Nxt},1}(t))\right\}_{t \in \mathbb{F}_q, j \in [m-1]}$ and evaluations $\left\{b_t = \hat{x}(C^{\mathsf{Prv},1}(w_i, R_t))\right\}_{t \in [|R|]}$ and $i^* = i$ and oracle access to $P^{(i)}$, it outputs $\left\{\hat{x}(C^{\mathsf{Nxt},1}(t))\right\}_{t \in \mathbb{F}_q}$.*

2. **Learning $C^{\mathsf{Nxt},2}(t) \triangleq A^k \cdot C_2(t)$ from $C^{\mathsf{Prv},2}(t) \triangleq A^k \cdot C_1(t)$.** *When we give $C_{\mathsf{LrnNext},i}$ the points $\left\{a_t^{(j)} = \hat{x}(A^{-j\cdot q^i} \cdot C^{\mathsf{Nxt},2}(t))\right\}_{t \in \mathbb{F}_q, j \in [m-1]}$ and evaluations $\left\{b_t = \hat{x}(C^{\mathsf{Prv},2}(w_v, R_t))\right\}_{t \in [|R|]}$ and $i^* = v$ and oracle access to $P^{(i)}$, it outputs $\left\{\hat{x}(C^{\mathsf{Nxt},2}(t))\right\}_{t \in \mathbb{F}_q}$.*

[30] The bound $\mathrm{poly}(1/\rho) \leq q^{10v}$ assumes that $m \leq n^\zeta$ for a universal constant $\zeta > 0$. We can indeed assume this without loss of generality, otherwise Theorem 5.1 is trivial.

Proof. Let $C\colon \mathbb{F}_q \to \mathbb{F}_q^v$ be a degree-d^{crv} curve sampled by Corollary 5.14 with $\varepsilon = \rho/12$ and $\delta = q^{-30v}$, and let $R \subseteq \mathbb{F}_{q_0}$ be a set sampled by Corollary 5.15 with $\delta = q^{-30v}$. Denote $r = |R|$. We let $C_1 = C$, and define $C_2\colon \mathbb{F}_q \to \mathbb{F}_q^v$ as the unique curve of degree $(r \cdot q_0 - 1)$ satisfying the following:

$$\forall a \in \{0,...,q_0-1\}, \forall u \in R, \quad C_2(w_a,u) = S_a \cdot C_1(w_a,u), \tag{5.5}$$
$$\text{where } S_a = \begin{cases} A^{q^a} & a \in \{0,...,v-1\} \\ \mathsf{Id} & a \in \{v,...,q_0-1\} \end{cases}$$

Observe that $\max\{\deg(C_1),\deg(C_2)\} < d^{\mathsf{crv}} \cdot q_0 \cdot r = D$, and that we can sample the curves and R and print them in space $O(\log n)$ and by using $O(\log n)$ coins.

Thus, we turn to the analysis. We first claim that the two curves have sufficient "sampling" properties and "intersection sampling" properties, as follows.

Claim 5.19.1 (each curve is a sampler, marginally)**.** For any fixed $k \in [q^v - 1]$, when choosing C from Corollary 5.14 with $\varepsilon = \rho/12$ and $\delta = q^{-30v}$ and R from Corollary 5.15 with $\delta = q^{-30v}$,

1. The distribution $A^k \cdot C_1$ is an (ε,δ)-sampler.
2. The distribution $A^k \cdot C_2$ is an (ε',δ')-sampler, where $\varepsilon' = 3\varepsilon = \rho/4$ and $\delta' = 2\delta \cdot q_0 < q^{-20v}$.

Proof. First observe that C_1 is an (ε,δ)-sampler, because $C_1 = C$. Next, for any shift A^k, the curve $A^k \cdot C_1$ is also an (ε,δ)-sampler; this is since A^k is invertible, and so the mapping of the image of C_1 to the image of $A^k \cdot C_1$ is a bijection.

We prove that C_2 is a sampler relying on the fact that C is a strong sampler and on the fact that R is a sampler. Specifically, fix any choice of $C = C_1$. We first claim that for every $T \subseteq \mathbb{F}_q^v$ and every fixed $a \in \{0,...,q_0-1\}$, with probability at least $1-\delta$ over R it holds that

$$\left| \Pr_{u\in R}[C_2(w_a,u) \in T] - \Pr_{u\in\mathbb{F}_{q_0}^{\Delta-1}}[S_a \cdot C_1(w_a,u) \in T] \right| \le \varepsilon, \tag{5.6}$$

and

$$\left| \Pr_{u\in R}[C_2(w_a,u) \in T] - \Pr_{u\in\mathbb{F}_{q_0}^{\Delta-1}}[C_2(w_a,u) \in T] \right| \le \varepsilon. \tag{5.7}$$

Indeed, the statements in Eqs. (5.6) and (5.7) are true because R is an (ε,δ)-sampler in $\mathbb{F}_{q_0}^{\Delta-1}$ (and considering the tests $T_{w_a}(u) = \mathbf{1}[S_a \cdot C_1(w_a,u) \in T]$ and $T'_{w_a}(u) = \mathbf{1}[C_2(w_a,u) \in T]$).

It follows that for every fixed C_1 and $T \subseteq \mathbb{F}_q^v$, with probability at least $1 - \delta \cdot q_0$ over R we have

$$\begin{aligned}
&\left| \Pr_{a\in\{0,...,q_0-1\},u\in\mathbb{F}_{q_0}^{\Delta-1}}[C_2(w_a,u) \in T] - \Pr_{a\in\{0,...,q_0-1\},u\in\mathbb{F}_{q_0}^{\Delta-1}}[S_a \cdot C_1(w_a,u) \in T] \right| \\
&= \left| \mathop{\mathbb{E}}_{a\in\{0,...,q-1\}} \left[\Pr_{u\in\mathbb{F}_{q_0}^{\Delta-1}}[C_2(w_a,u) \in T] - \Pr_{u\in\mathbb{F}_{q_0}^{\Delta-1}}[S_a \cdot C_1(w_a,u) \in T] \right] \right| \\
&\le \mathop{\mathbb{E}}_{a\in\{0,...,q-1\}} \left[\left| \Pr_{u\in\mathbb{F}_{q_0}^{\Delta-1}}[C_2(w_a,u) \in T] - \Pr_{u\in\mathbb{F}_{q_0}^{\Delta-1}}[S_a \cdot C_1(w_a,u) \in T] \right| \right] \\
&\le 2\varepsilon.
\end{aligned} \tag{5.8}$$

Now, consider the joint distribution (C_1,R,C_2), which is obtained by choosing $C = C_1$ and R independently, and defining $C_2 = C_2(C_1,R)$ as in Eq. (5.5). Assume towards a contradiction

that there is $T \subseteq \mathbb{F}_q^v$ such that with probability more than $2\delta \cdot q_0$ over (C_1, R, C_2) it holds that $\Pr_{t\in\mathbb{F}_q}[C_2(t) \in T] - |T|/q^v > 3\varepsilon$.[31]

Now, by Eq. (5.8), for every fixed choice of C_1, with probability $1 - \delta \cdot q_0$ over R we have that

$$\Pr_{a\in\{0,\ldots,q_0-1\},u\in\mathbb{F}_{q_0}^{\Delta-1}}[S_a \cdot C_1(w_a, u) \in T] - |T|/q^v > \varepsilon. \tag{5.9}$$

We call every choice of R satisfying Eq. (5.9) good for C_1.

Next, consider the following test $T' \subseteq \mathbb{F}_q \times \mathbb{F}_q^v$. Given $(t, \vec{z})$, we parse $t = (w_a, u) \in \mathbb{F}_{q_0} \times \mathbb{F}_{q_0}^{\Delta-1}$, and define $M((w_a, u), \vec{z}) \triangleq S_a \cdot \vec{z} \in \mathbb{F}_q^v$; we include $(t, \vec{z})$ in T' iff $M(t, \vec{z}) \in T$. Note that when $(t, \vec{z})$ is chosen uniformly, the distribution $M(t, \vec{z})$ is uniform in $\mathbb{F}_q^v$, and hence

$$\Pr_{t,\vec{z}\in\mathbb{F}_q\times\mathbb{F}_q^v}[(t, \vec{z}) \in T'] = |T|/q^v.$$

On the other hand, when $(t, \vec{z})$ is uniformly chosen from the set $\{(t, C_1(t))\}_{t\in\mathbb{F}_q}$, the distribution $M(t = (w_a, u), \vec{z})$ is the uniform distribution on $\{S_a \cdot C_1(w_a, u)\}_{a\in\{0,\ldots,q_0-1\},u\in\mathbb{F}_{q_0}^{\Delta-1}}$, and hence

$$\Pr_{t\in\mathbb{F}_q}\left[(t, C_1(t)) \in T'\right] = \Pr_{a\in\{0,\ldots,q_0-1\},u\in\mathbb{F}_{q_0}^{\Delta-1}}\left[S_a \cdot C_1(w_a, u) \in T\right].$$

Plugging the two equations above into Eq. (5.9), whenever R is good for C_1, we have that

$$\Pr_{t\in\mathbb{F}_q}\left[(t, C_1(t)) \in T'\right] - \Pr_{t,\vec{z}\in\mathbb{F}_q\times\mathbb{F}_q^v}[(t, \vec{z}) \in T'] > \varepsilon. \tag{5.10}$$

It follows that

$$\begin{aligned}\Pr_{C_1}[\text{Eq. (5.10) holds}] &\geq \Pr_{C_1,R}\left[\left(\Pr_{t\in\mathbb{F}_q}[C_2(t) \in T] - |T|/q^v > 3\varepsilon\right) \wedge R \text{ is good for } C_1\right] \\ &\geq 1 - 2\delta \cdot q_0 - \delta \cdot q_0,\end{aligned}$$

contradicting the fact that $C = C_1$ is a strong (ε, δ)-sampler. □

Claim 5.19.2 (interleaved curves agree on the points specified by R)**.** For any fixed $i \in \{0, ..., v-1\}$ and any choice of C and R we have that

$$R \subseteq \left\{u \in \mathbb{F}_{q_0}^{\Delta-1} : A^{q^i} \cdot C_1(w_i, u) = C_2(w_i, u)\right\}.$$

Moreover, for any fixed $k \in [q^v - 1]$, the claim still holds if we simultaneously replace "C_1" by "$A^k \cdot C_1$" and "C_2" by "$A^k \cdot C_2$".

Proof. The basic claim follows from the definition of C_2, and the "moreover" part follows immediately from the basic claim. □

Now, for Item (1) of our lemma, fix i and k, and let $C^{\mathsf{Nxt},1}(t) = A^{k+q^i} \cdot C_1(t)$. By Lemma 5.17, with probability at least $1 - q^{-10v}$ over $C^{\mathsf{Nxt},1}$ and R, when we give $C_{\mathsf{LrnNext},i}$ the points

$$\left\{a_t^{(j)} = \hat{x}(A^{-j\cdot q^i} \cdot C^{\mathsf{Nxt},1}(t))\right\}_{t\in\mathbb{F}_q, j\in[m-1]}$$

[31] Note that if there is T' such that $\left|\Pr_{t\in\mathbb{F}_q}[C_2(t) \in T'] - |T'|/q^v\right| > 3\varepsilon$ with probability more than $2\delta \cdot q_0$, then there is T such that $\Pr_{t\in\mathbb{F}_q}[C_2(t) \in T'] - |T'|/q^v > 3\varepsilon$ with probability more than $\delta > 0$ (i.e., by taking either $T = T'$ or T as the complement of T'). Thus, to rule out the former it suffices to rule out the latter.

and the evaluations $\left\{b_t = \hat{x}(C^{\mathsf{Nxt},1}(w_i, R_t))\right\}_{t\in[|R|]}$ and $i^* = i$ and oracle $P^{(i)}$, it outputs $\left\{C^{\mathsf{Nxt},1}(t)\right\}_{t\in\mathbb{F}_q}$, as long as the distributions over $C^{\mathsf{Nxt},1}$ and R are appropriate samplers. By Claim 5.19.1, the distribution over $C^{\mathsf{Nxt},1}$ is indeed such a sampler. By Claim 5.19.2, the curves $C^{\mathsf{Nxt},1}$ and $C^{\mathsf{Prv},1}(t) = A^k \cdot C_2(t)$ agree on the point-set $\{(w_i, R_t)\}_{t\in[r]}$, and hence the functionality of $C_{\mathsf{LrnNext},i}$ remains identical if we replace the set of b_t's above by $\left\{b_t = \hat{x}(C^{\mathsf{Prv},1}(w_i, R_t))\right\}_{t\in[r]}$.

For Item (2), similarly, we fix i, k, and let $C^{\mathsf{Nxt},2}(t) = A^k \cdot C_2(t)$. We use Lemma 5.17 identically to the proof above, the only difference being that now we argue about $C_{\mathsf{LrnNext},i}$ when it is given $i^* = v$ (rather than $i^* = i$). The claim follows relying on the fact that $C^{\mathsf{Nxt},2}$ is an appropriate sampler (by Claim 5.19.1) and that $C^{\mathsf{Nxt},2}$ and $C^{\mathsf{Prv},2}(t) = A^k \cdot C_1(t)$ agree on the point-set $\{(w_v, u)\}_{u\in\mathbb{F}_{q_0}}$ (by the definition of C_1 and C_2 on $\{(w_v, \cdot)\}$, in Eq. (5.5)).

The two paragraphs above established that for every fixed i, k, the statements in Items (2) and (1) hold with probability at least $1 - q^{-10v}$. By a union-bound over $i \in \{0, ..., v-1\}$ and $k \in [q^v - 1]$, with probability at least $1 - q^{-5v}$ the two statements hold for every i, k. □

5.4.4 The main reconstruction algorithm

We are now ready to state and prove the main algorithm of the current section. This algorithm uses $O(\log n)$ coins and $O(\log n)$ space, makes queries to $\hat{x}$, and with high probability prints $\vec{v} \in \mathbb{F}_q^v$ and a circuit that computes the mapping $y \mapsto \hat{x}(A^y \cdot \vec{v})$.

Proposition 5.20. *There is an algorithm that gets input 1^n, uses $O(\log n)$ random coins and $O(\log n)$ space, makes $O(m \cdot q)$ queries to $\hat{x}$,[32] and for every collection of $(\rho/2)$-good predictors $\vec{P} = P^{(0)}, ..., P^{(v-1)}$, with probability $1 - q^{-O(v)}$ it prints a circuit $R_0 \colon \{0, ..., q^v - 1\} \to \mathbb{F}_q$ of size $\mathrm{poly}(m, \log(n))$ such that $R_0^{\vec{P}}(y) = \hat{x}(A^y \cdot \vec{1})$.*

Proof. We first describe the circuit R_0 and then explain how to construct it. The circuit has hard-wired choices of curves C_1, C_2 and a set $R \subseteq \mathbb{F}_{q_0}^{\Delta-1}$, as well as the circuits $C_{\mathsf{LrnNext},i}$ from Lemma 5.19 with all values of $i \in \{0, ..., v-1\}$. Let $\vec{v} = C_1(1)$.

The circuit receives $y \in \{0, ..., q^v - 1\}$ and parses it in basis q as $y = \sum_{i=0}^{v-1} y_i \cdot q^i$. Denote $y^{(-1)} = 0$, and for $i \in \{0, ..., v-1\}$, let $y^{(i)} = \sum_{i'=0}^{i} y_{i'} \cdot q^{i'}$. The circuit works in v iterations, where in iteration $i \in \{0, ..., v-1\}$ it has already obtained the values

$$\left\{\hat{x}\left(A^{y^{(i-1)}+j\cdot q^i} \cdot C_b(t)\right)\right\}_{t\in\mathbb{F}_q, b\in\{1,2\}, j\in[m-1]}$$

and its goal is to compute the values of

$$\left\{\hat{x}\left(A^{y^{(i-1)}+j'\cdot q^i} \cdot C_b(t)\right)\right\}_{t\in\mathbb{F}_q, b\in\{1,2\}, j'\in\{m,...,(m-1)\cdot q\}},$$

where we denote $A^0 = \mathsf{Id}$ and $\sum_{i'=0}^{-1} y_{i'} \cdot q^{i'} = 0$.

The values $\left\{\hat{x}\left(A^j \cdot C_b(t)\right)\right\}_{t\in\mathbb{F}_q, b\in\{1,2\}, j\in[m-1]}$ will be hard-wired into R_0, so the first iteration has the values it needs to start its execution. Observe that after iteration i completes successfully, the circuit has the values that it needs for iteration $i+1$. Also note that after iteration $v-1$ the circuit has learned the value $\hat{x}\left(A^y \cdot C_1(1)\right)$, as we wanted.

Thus, it remains to describe how a single iteration i is executed. The circuit R_0 uses the circuit $C_{\mathsf{LrnNext},i}$. For $j' \in \{m, ..., (m-1)\cdot q\}$, we first use Item (1) of Lemma 5.19 with value $k = y^{(i-1)} + (j'-1)\cdot q^i$ and then use Item (2) of Lemma 5.19 with value $k = y^{(i-1)} + j' \cdot q^i$. In both cases, the circuit R_0 gives $C_{\mathsf{LrnNext},i}$ access to its oracle $P^{(i)}$.

[32] The number of queries can be reduces to $O(m \cdot d \cdot D)$, since the algorithm just needs to learn the values of $2m$ degree-$d \cdot D$ polynomials. However, for simplicity we do not optimize the number of queries.

Tedious verification, which may be skipped. To carefully verify the correct use of Lemma 5.19, for each $j' \in \{m, ..., m \cdot q\}$, denote $C^{\mathsf{Nxt},j',1}$ as $C^{\mathsf{Nxt},1}$ when we use Item (1), and denote $C^{\mathsf{Nxt},j',2}$ as $C^{\mathsf{Nxt},2}$ when we use Item (2); analogously, denote $C^{\mathsf{Prv},j',1}, C^{\mathsf{Prv},j',2}$.

Now, fix $j' \in \{m, ..., m \cdot q\}$, and recall that we enter step j' (of iteration i) having already learned $\left\{\hat{x}(A^{y^{(i-1)}+j\cdot q^i} \cdot C_b(t))\right\}_{t,b,j\in[j'-1]}$ in previous steps (or in iteration $i-1$). To reduce notational clutter, for a curve $C\colon \mathbb{F}_q \to \mathbb{F}_q^v$ we will denote $\hat{x}(C) = \{\hat{x}(C(t))\}_{t\in\mathbb{F}_q}$. We also use the shorthand notation $\vec{a}^{(j)} = (a_t^{(j)})_{t\in\mathbb{F}_q}$.

- When we use Item (1), we have $C^{\mathsf{Nxt},j',1} = A^{y^{(i-1)}+j'\cdot q^i}\cdot C_1$ and $C^{\mathsf{Prv},j',1} = A^{y^{(i-1)}+(j'-1)\cdot q^i}\cdot C_2$. The evaluations we learned going into step j' include $\left\{\vec{a}^{(j)} = \hat{x}(A^{-j\cdot q^i}\cdot C^{\mathsf{Nxt},j',1})\right\}_{j\in[m-1]}$. Also, in the previous step $j'-1$ we learned $\hat{x}(C^{\mathsf{Nxt},j'-1,2}) = \hat{x}(A^{y^{(i-1)}+(j'-1)\cdot q^i}\cdot C_2)$, so in particular we learned $\left\{b_t = \hat{x}(C^{\mathsf{Prv},j',1}(w_i, R_t))\right\}_{t\in[r]}$.
- When we use Item (2), we have $C^{\mathsf{Nxt},j',2} = A^{y^{(i-1)}+j'\cdot q^i}\cdot C_2$ and $C^{\mathsf{Prv},j',2} = A^{y^{(i-1)}+j'\cdot q^i}\cdot C_1$. The evaluations we learned going into step j' include $\left\{\vec{a}^{(j)} = \hat{x}(A^{-j\cdot q^i}\cdot C^{\mathsf{Nxt},j',2})\right\}_{j\in[m-1]}$, and in the most recent usage of Item (1) we learned $\hat{x}(C^{\mathsf{Nxt},j',1}) \supseteq \left\{b_t = \hat{x}(C^{\mathsf{Prv},j',2}(w_v, R_t))\right\}_{t\in[r]}$.

The execution of the two items yields the values $\hat{x}(A^{y^{(i-1)}+j'\cdot q^i}\cdot C_b)$ for $b \in \{1,2\}$, so we can continue to step $j'+1$.

Complexity and success probability. Note that R_0 has $O(m\cdot q)$ elements of $\mathbb{F}_q$ hard-wired into it, as well as two curves (i.e., $2q$ elements of $\mathbb{F}_q^v$), a set of size r, and v circuits of size $\mathrm{poly}(q, 1/\rho)$. For its execution, it works in v iterations, and in each iteration it simulates $C_{\mathsf{LrnNext},i}$ and stores $O(m\cdot q^2)$ values. Thus, overall, R_0 can be implemented in size $\mathrm{poly}(m, \log(n))$.

Moreover, since the functionality of R_0 (given the hard-wired information) is can be implemented by a uniform machine, the following holds: There is a (uniform) Turing machine that gets as input the information that is supposed to be hard-wired into R_0 (i.e., the elements of $\mathbb{F}_q$ for the first iteration, the two interleaved curves, the sampled set of size r, and the v circuits $C_{\mathsf{LrnNext},i}$) as well as an input y, and computes the value of the corresponding circuit R_0 (i.e., the R_0 that is obtained by the given "hard-wired" information) at y in time $\mathrm{poly}(m, \log(n))$. Thus, similarly to the proof of Lemma 5.17, observe that the foregoing functionality can be computed by an $O(\log(m, \log(n)))$-space-uniform circuit family $\left\{R'_{0,n}\right\}_{n\in\mathbb{N}}$ of size $\mathrm{poly}(m, \log(n))$.

The machine M that prints R_0 simulates the machine that prints $R'_0 = R'_{0,n}$, which uses space $O(\log(m, n\log(n))) \leq O(\log(n))$, and hard-wires the needed information into the corresponding input gates of R'_0. Specifically, M samples C_1, C_2, R using Lemma 5.19, queries $\hat{x}$ at points $\left\{A^j \cdot C_b(t)\right\}_{j\in[m-1],t\in\mathbb{F}_q,b\in\{0,1\}}$, and hard-wires all of this information into the corresponding input gates for R'_0. (Recall that M can compute powers of A in space $O(\log n)$, by Proposition 5.7, and that M can evaluate C_b at any point t in space $O(\log n)$, by Lemma 5.19.) In addition, the machine M computes the descriptions of $C_{\mathsf{LrnNext},i}$ for all $i \in \{0, ..., v-1\}$, using Lemma 5.17, and hard-wires them into the corresponding input gates of R'_0. Thus, M runs in space $O(\log n)$, and the circuit that it prints computes the mapping $y \mapsto R_0(y)$.

Note that with probability $1 - q^{-O(v)}$ over the randomness of M (which was used only for the algorithm of Lemma 5.19), for all $y \in [q^v - 1]$, the circuits $C_{\mathsf{LrnNext},i}$ succeed in all the $m\cdot q\cdot v < q^v$ iterations when R_0 executes on input y. Hence, with probability at least $1 - q^{-O(v)}$, the machine M prints $\vec{v}$ and R_0 that succeeds in computing $y \mapsto A^y \cdot \vec{v}$ on all $y \in [q^v - 1]$. □

5.5 Putting It All Together: The Reconstruction Procedure

Our goal now is to prove the reconstruction part of Theorem 5.1. That is, we show an algorithm RSU that gets input 1^n, and gets oracle access to $x \in \{0,1\}^n$ and to $(1/m^2)$-next-bit-predictors $\left\{P_i \colon \{0,1\}^{j_i} \to \{0,1\}\right\}_{i \in \{0,\ldots,v\}}$, runs in space $O(\log n)$, uses $O(\log n)$ random coins, and prints an oracle circuit $C \colon \{0,1\}^{\log(n)} \to \{0,1\}$ of size $\mathrm{poly}(m)$ such that with probability at least $1/\mathrm{poly}(m)$ it holds that $C^{P_0,\ldots,P_v}(u) = x_u$ for all $u \in [n]$.

Discrete log. By Proposition 5.9, we can print in space $O(\log n)$ and with $O(\log n)$ random coins an oracle circuit C_{dl} of size $\mathrm{poly}(m, \log(n))$ such that $C_{\mathsf{dl}}^{P_v}(A^y \cdot \vec{1}) = y$ for all $y \in \{0,1\}^{v \cdot \log(q)}$. The success probability for printing C_{dl} is at least $1/\mathrm{poly}(m)$.

q-ary reconstruction. By Proposition 5.20, given oracle access to $\hat{x}$, we can print in space $O(\log n)$ and with $O(\log n)$ random coins a circuit $R_0 \colon [q^v - 1] \to \mathbb{F}_q$ of size $\mathrm{poly}(m, \log(n))$ such that $R_0^{\vec{P}}(y) = \hat{x}(A^y \vec{1})$, when $\vec{P}$ is a sequence of $(\rho/2)$-good predictors for the $L_i^{(q)}$'s. The queries to $\hat{x}$ can be answered in space $O(\log n)$, given our oracle access to x, and the success probability of this algorithm is high (i.e., $1 - q^{-O(v)} > 1/2$).

List-decoding. Now, the list-decoder for $\mathsf{Enc}_{\mathsf{STV}}$ from Theorem 5.8 runs in time $\mathrm{poly}(m, \log(n))$, and hence a circuit $\mathsf{Dec}_{\mathsf{STV}}$ of such size implementing its functionality can be printed in space $O(\log(m, \log(n))) = O(\log n)$. By a standard argument (see, e.g., [SU05, Lemma 4.16], following [TZS06]), given oracle access to a $(1/m^2)$-next-bit-predictor $P_i \colon \{0,1\}^{j_i} \to \{0,1\}$ for L_i, we can compute a $(\rho/2)$-good predictor $P_i^{(q)} \colon \mathbb{F}_q^{j_i} \to \mathbb{F}_q^{\bar{\rho}}$ for $L_i^{(q)}$ as follows:

- Given $w_1, \ldots, w_{j_i} \in \mathbb{F}_q$, for each $k \in [\ell_q]$, compute $r_k = P_i(\mathsf{Enc}_{\mathsf{STV}}(w_1)_k, \ldots, \mathsf{Enc}_{\mathsf{STV}}(w_{i_j})_k)$.
- Let $r = (r_1, \ldots, r_{\ell_q})$, and output the list of decoded messages that $\mathsf{Dec}_{\mathsf{STV}}$ outputs when given access to the corrupt codeword r.

Observe that we can implement $P_i^{(q)}$ by an oracle circuit $C_j^{(\mathsf{Dec})}$ of size $\mathrm{poly}(m, \log(n))$ (which gets oracle access to P_i), and that this circuit can be constructed in space $O(\log n)$.

Combining the ingredients. We print an oracle circuit $C_{\hat{x}}$ that computes $\hat{x}$, as follows:

- Given $u \in \mathbb{F}_q^v$, use $C_{\mathsf{dl}}(u)$ to compute $y \in \{0,1\}^{v \cdot \log(q)} \equiv [q^v - 1]$ such that $u = A^y \cdot \vec{1}$. (Note that for every u there exists such y, since A is a generator matrix.)
- Use $R_0(y)$ to compute $\hat{x}(A^y \cdot \vec{1}) = \hat{x}$. Whenever R_0 queries one of its $(\rho/2)$-good predictors, answer using $C_j^{(\mathsf{Dec})}$ and our oracle access to the next-bit-predictors.

Observe that the size of the circuit $C_{\hat{x}}$ is at most $\mathrm{poly}(m, \log(n))$, and that it can be printed in space $O(\log n)$ and with $O(\log n)$ random coins, with success probability $1/\mathrm{poly}(m)$.

The final circuit C needs to compute the mapping $u \mapsto x(u)$. Recall that u represents the coefficient of some monomial in $\hat{x} \colon \mathbb{F}_q^v \to \mathbb{F}_q$, say $y_1^{e_1} \cdot y_2^{e_2} \cdot \ldots \cdot y_v^{e_v}$ where $\sum_{k \in [v]} e_k \leq d$. The coefficient of this monomial is determined by the evaluation of $\hat{x}$ of at most d points, and thus the circuit C invokes $C_{\hat{x}}$ for d times and outputs the corresponding linear combination.

Remark 5.21. An interesting feature of the reconstruction is that it can be split into two parts, where one part succeeds with high probability, and the other succeeds with low probability but

produces a circuit that never outputs the wrong answer. Specifically, the only part that succeeds with low probability is the reconstruction for L_v from Proposition 5.9, but the output circuit of this procedure can test whether or not it correctly computed discrete log.

6 Proof of the Main Theorems

6.1 A New Bootstrapping System, and the Main Pair of Algorithms

6.1.1 A Bootstrapping System Based on Reachability

Before proving our main theorems, we first define the key bootstrapping system:

Theorem 6.1 (Reachability Bootstrapping System)**.** *There is an $n \times n$ bootstrapping system with the following properties. Let G be an arbitrary graph on n vertices. Let $P_i = P_i(G) \in \{0,1\}^{n^2}$ be defined as:*

$$(P_i)_{s,t} = \mathbb{I}\,[\textit{There is a path of length at most } i \textit{ from } s \textit{ to } t.]$$

Then the following hold:

1. ***Layer DSR.*** *There is a space $O(\log n)$ algorithm DSR such that $\mathsf{DSR}^{G,P_i}(i+1)$ outputs P_{i+1}.*
2. ***Base Case.*** *Layer 0 is computable in space $O(\log n)$ with oracle access to G.*
3. ***Layer $\mathbf{NL}$ Computability.*** *There is a nondeterministic logspace algorithm that on input G, i, s, t, computes $(P_i)_{s,t}$.*

Proof. For the first, consider computing $(P_{i+1})_{s,t}$ for arbitrary s, t. A path of length $i+1$ from $s \to t$ exists if and only if there is a vertex v such that there is a path of length i from $s \to v$, and (v,t) is an edge in G. Then by enumerating over v and using queries to P_i and G, it is easy to see the DSR algorithm can compute $(P_{i+1})_{s,t}$. The second item is direct for the same reason. For the final component, note that

$$L_Y = \{(G,s,t,i) : \text{ there exists an } s \to t \text{ path of length at most } i \text{ in } G\}$$

is clearly in $\mathbf{NL}$, and

$$L_N = \{(G,s,t,i) : \text{ there does not exist an } s \to t \text{ path of length at most } i \text{ in } G\}$$

is clearly in $\mathbf{coNL}$, and by [Imm88, Sze88] likewise lies in $\mathbf{NL}$. Then our $\mathbf{NL}$ machine interprets the first bit of the guess tape as Y or N, and then runs the corresponding $\mathbf{NL}$ verifier. □

We require one more lemma, combining the D2P for Nisan's generator with the generator of Theorem 5.1.

Lemma 6.2 (single layer reconstruction)**.** *There are a pair of algorithms GEN and REC that together work as follows. For a graph G of size n and $f \in \{0,1\}^{n^2}$,*

- $\mathsf{GEN}(G, f)$ *runs in space $O(\log n)$ and either returns $\vec{h}$ such that $T_G(\vec{h}) = 1$, where T_G is defined as in Theorem 4.8, or $\perp$.*
- $\mathsf{REC}(G, f)$ *runs in space $\mathrm{polylog}(n)$ and time $\mathrm{poly}(n)$, and if $\mathsf{GEN}(G, f) = \perp$, then REC prints an oracle Turing machine C of description size $v(n) = \mathrm{polylog}(n)$ that runs in space $O(v(n))$, and satisfies $C^G = f$.*

Moreover, REC *can alternately output a circuit* C *of size* $v(n)$ *and the (constant-size) description of an oracle machine* M *that runs in space* $O(\log n)$ *so that* $\mathsf{tt}(C^{M^G}) = f$.

Proof. First, let SU be the generator of Theorem 5.1 with $N = n^2$ and $M = s(n) \leq (n^2)^{\varepsilon_{\mathsf{SU}}}$ where $s(n)$ is the seed length of T_G (Theorem 4.8), such that SU takes as input a truth table of length n^2, and outputs lists

$$L_1, \ldots, L_\ell$$

for $\ell = O(\log n)$, where L_i is a list of strings of length $s(n)$. Note that SU can be computed in space $O(\log n)$.

The algorithm GEN**.** The algorithm GEN simply computes the output of SU^f and returns the first element of the lists such that $T_G(\vec{h}) = 1$ (recall that T_G can be evaluated in logspace), and if no such element exists, GEN returns $\perp$.

The algorithm REC**.** We now define the algorithm REC. In the case that $\mathsf{GEN}(G, f) = \perp$, we have for every list L_i,

$$\left| \mathop{\mathbb{E}}_{y \leftarrow L_i} \left[T_G(y) \right] - \mathbb{E}\left[T_G(\mathbf{U}) \right] \right| = \mathbb{E}\left[T_G(\mathbf{U}) \right] \geq 1 - o(1),$$

where the inequality follows from Item 3. In particular, we have that T_G is a $(1/2)$-distinguisher for every L_i. Recall that

$$(\mathsf{PRED}_1, \ldots, \mathsf{PRED}_{\mathrm{polylog}(n)})$$

is a $\delta = 1/2$ to $\rho = \Omega(1/\log^2 n)$-D2P transformation for T_G, and that given G, the predictors can be evaluated in logspace. We first determine which predictor obtains good advantage for each list, using that each predictor can be evaluated in logspace.

Claim 6.3. *There is a space* $\mathrm{poly}(\log n)$*, time* $\mathrm{polylog}(n)$ *algorithm that prints a list* $K = (k_1, \ldots, k_\ell)$ *such that for every* i*,* PRED_{k_i} *is a* ρ*-predictor for* L_i*.*

We then store this list K on the worktape in space $\mathrm{polylog}(n)$ (and will hardwire it into the returned circuit). Next, we call the algorithm RSU of Theorem 5.1 with $f = f$ and predictors

$$\mathsf{PRED}_K = (\mathsf{PRED}_{k_1}, \ldots, \mathsf{PRED}_{k_\ell}).$$

We enumerate over all $O(\log n)$ random coins used by RSU until we find a set of coins on which it prints an oracle circuit C' which outputs f when given the predictors as oracles (for this test, we use that we can evaluate the predictors in logspace with oracle access to G). This circuit is of size $v' = \mathrm{polylog}(n)$ and satisfies

$$\mathsf{tt}(C'^{\mathsf{PRED}_{k_1}, \ldots, \mathsf{PRED}_{k_\ell}}) = f.$$

Finally, we let C be the machine that evaluates C', and answers oracle queries to the predictors using the list K and the machine implementing the D2P transform for T_G (for which C provides oracle access to G), which has constant size given the predictor indices and can be evaluated in space $O(\log n)$. By choosing $v(n) = \mathrm{polylog}(n)$ large enough, we have that the total description size of C is $v(n)$ and C can be evaluated in space v, as claimed.

For the moreover claim, we let the machine M be the space $O(\log n)$ machine of Theorem 4.8 that on (G, i, x) returns $\mathsf{PRED}_i(x)$, where we give the machine oracle access to G. □

6.1.2 The pair of algorithms

We can now prove our main algorithmic result. We state it in a way that can be easily used to imply both Theorem 1 and Theorem 2.6.

Theorem 6.4. *There are algorithms $\mathcal{A}_2, \mathcal{A}_1$ such that for every pair of graphs G_1, G_2 on n vertices, at least one of the following holds:*

1. *$\mathcal{A}_1(G_1, G_2)$ outputs the transitive closure of G_1. Moreover, $\mathcal{A}_1$ runs in space $\mathrm{polylog}(n)$ and time $\mathrm{poly}(n)$.*
2. *$\mathcal{A}_2(P(G_1), G_2)$ outputs $\widetilde{\mathbf{G}}$ such that*

$$\left\| \widetilde{\mathbf{G}} - \mathbf{G}_2^n \right\| \leq 1/n^2$$

where $P(G_1) = P_0, \ldots, P_n$ are the layers of the bootstrapping system of Theorem 6.1. Moreover, $\mathcal{A}_2$ runs in space $O(\log n)$.

Proof. Let

$$T \stackrel{\text{def}}{=} T_{G_2} \colon \{0,1\}^{s(n)=O(\log^2 n)} \to \{0,1\}$$

be the test function of Theorem 4.8, and recall that the function can be computed in space $O(\log n)$ given read-only access to G_2 and the input. Next, we instantiate the bootstrapping system of Theorem 6.1 with $G = G_1$, and denote the layers by $P_0, \ldots, P_n$.

We say a layer i is $\mathsf{compressible}$ if the algorithm $\mathsf{GEN}(G_2, P_i)$ of Lemma 6.2 returns $\perp$ (i.e., does not produce a good hash function); recall that GEN is evaluable in logspace.

Finally, we define both algorithms:

- The algorithm $\mathcal{A}_2(P(G_1), G_2)$ works as follows. It iterates over $i = 0, \ldots, n$ (and recall that we are given $P_0, \ldots, P_n$ as input). Fixing a current layer i, we run $\mathsf{GEN}(G_2, P_i)$ and determine if it outputs $\perp$. If so, we increment i and move to the next layer, and if $i = n$ we abort and return $\perp$.

  Otherwise, GEN returns $\vec{h}$ so that $T(\vec{h}) = 1$ (take $\vec{h}$ from the smallest such i). From Item 2, we have that $\widetilde{\mathbf{G}} = \mathbf{G}_{t,\vec{h}}$ (which we can compute in space $O(\log n)$ with access to $\vec{h}$) satisfies

$$\left\| \widetilde{\mathbf{G}} - \mathbf{G}_2^n \right\| \leq n^{-2}$$

  so the output is as required.

- The algorithm $\mathcal{A}_1(G_1, G_2)$ is a deterministic $\mathbf{SC}$ algorithm that builds a small (oracle) machine for P_i layer by layer. Our inductive claim is as follows:

  Claim 6.5. *There exists an oracle Turing machine C of description size $v(n) = \mathrm{polylog}(n)$ that runs in space $O(v)$ such that*

$$\mathsf{tt}\left(C^{G_2}\right) = P_i.$$

  We initialize $i = 0$, and note that there is a very simple machine of size $O(\log n)$ that computes P_0 and hence we satisfy the base case. Fixing the current layer i, assume we have such a machine C for P_{i-1} whose encoding is kept on the work tape. First note that by composing this machine with the DSR algorithm, Item 1 of Theorem 6.1, we can compute P_i in space $O(v)$.

Next, we run the algorithm $\mathsf{GEN}(G_2, P_i)$ of Lemma 6.2 (answering queries to P_i using the DSR) and determine if it outputs a hash function. If so, we abort and return $\perp$. Otherwise, we run the algorithm $\mathsf{REC}(G_2, P_i)$, which returns a machine C such that $\mathsf{tt}(C^{G_2}) = P_i$, which we store on the work tape. Finally, we delete the machine for P_{i-1} and increment i. Note that C does not need oracle access to the prior machine, so the overall space consumption does not increase.

Finally, once we obtain such a machine for P_n (and have not aborted), we simply evaluate it and output the transitive closure. The runtime and space requirements follow from Lemma 6.2.

The fact that at least one such algorithm halts on every pair (G_1, G_2) follows directly from their definition. In particular, if every layer is compressible, we have that $\mathcal{A}_1$ will output a value, and otherwise $\mathcal{A}_2$ will output a value. □

6.1.3 Proofs of Theorem 1 and Theorem 2.6

We next give two instantiations of this result, formalizing Theorem 1 and Theorem 2.6. We begin with the former.

Theorem 6.6. *There are algorithms $\mathcal{A}_2, \mathcal{A}_1$ such that for every pair graphs G_1, G_2 on n vertices, at least one of the following holds:*

1. *$\mathcal{A}_1(G_1, G_2)$ computes the transitive closure of G_1. Moreover, $\mathcal{A}_1$ runs in space $\mathrm{polylog}(n)$ and time $\mathrm{poly}(n)$.*
2. *$\mathcal{A}_2(G_1, G_2)$ outputs $\widetilde{\mathbf{G}}$ such that*
$$\left\| \widetilde{\mathbf{G}} - \mathbf{G}_2^n \right\| \leq 1/n^2.$$
Moreover, $\mathcal{A}_2$ runs in nondeterministic space $O(\log n)$.

Proof. We apply the result of Theorem 6.4, and every time the algorithm $\mathcal{A}_2$ of that theorem requests a bit of P_i for some i, we use that the layers can be computed in nondeterministic space $O(\log n)$ via Theorem 6.1. □

Next, we formalize the first.

Theorem 6.7. *For every $\varepsilon > 0$, there are algorithms $\mathcal{A}_2, \mathcal{A}_1$ such that for every pair graphs G_1, G_2 on $m = 2^{\log^{1/2+\varepsilon/2} n}$ and n vertices respectively, at least one of the following holds:*

1. *$\mathcal{A}_1(G_1, G_2)$ computes the transitive closure of G_1. Moreover, $\mathcal{A}_1$ runs in space $\mathrm{polylog}(m)$ and time $\mathrm{poly}(n)$.*
2. *$\mathcal{A}_2(G_1, G_2)$ outputs $\widetilde{\mathbf{W}}$ such that*
$$\left\| \widetilde{\mathbf{G}} - \mathbf{G}_2^n \right\| \leq 1/n^2.$$
Moreover, $\mathcal{A}_2$ runs in space $O(\log^{1+\varepsilon} n)$.

Proof. We again apply the result of Theorem 6.4, with a further modification: we first pad G_1 to G_1' of size n by adding $n - m$ dummy vertices with 2 self-loops. When the algorithm $\mathcal{A}_2$ queries P_i for some i, queries corresponding to the dummy vertices are trivial, and other queries can be computed in nondeterministic space $O(\log m)$ via Theorem 6.1 (as the bootstrapping system is essentially on a graph of size m), and thus deterministic space $O(\log^{1+\varepsilon}(n))$ via Savitch [Sav70]. The algorithm $\mathcal{A}_1$ is unchanged (except that we perform the same padding). □

6.2 Scaled-Up Results

We next use the algorithms of Theorems 6.6 and 6.7 to prove the scaled-up tradeoffs. We begin with Theorem 2, and we now state a stronger technical version of the result:

Theorem 6.8 (stronger version of Theorem 2)**.** *For every constant $\varepsilon > 0$, at least one of the following holds:*

- $\mathbf{NSPACE}\left[m(n)\right] \subseteq i.o.\mathbf{TISP}\left[2^{O(m(n)^{2-\varepsilon})}, m(n)^{O(1)}\right]$, *for* $m(n) = n^{1/2+\varepsilon/2}$.
- $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}\left[O(n^{1+\varepsilon})\right]$.

Recall that in Theorem 2, the first item was stated with $m(n) = n$. This statement follows from Theorem 6.8 by a padding argument (i.e., if the first item of Theorem 6.8 holds with $m(n) = n^{1/2+\varepsilon/2}$, then by padding the same statement holds for $m(n) = n$).

The main lemma we will use to prove Theorem 6.8 is the following:

Lemma 6.9. *Let $\mathcal{B}$ be an arbitrary $\mathbf{BPSPACE}[n]$ machine and $\mathcal{N}$ be an arbitrary $\mathbf{NSPACE}[n^{1/2+\varepsilon/2}]$ machine. Then, there is a $\mathbf{SPACE}[n^{1+\varepsilon}]$ machine $\mathcal{S}$ and a $\mathbf{TISP}[2^{O(n)}, \mathrm{poly}(n)]$ machine $\mathcal{T}$ such that on every input length n, either $\mathcal{S}$ prints the truth table of $\mathcal{B}$ on inputs of length n, or $\mathcal{T}$ prints the truth table of $\mathcal{N}$ on input of length n.*

We first explain why this lemma implies Theorem 6.8. Suppose there exists such a $\mathcal{N}$ such that for every valid $\mathcal{B}$, the machine $\mathcal{S}$ prints a truth table on all but finitely many input lengths. In this case, we have $\mathbf{BPSPACE}[n] \subseteq \mathbf{SPACE}[n^{1+\varepsilon}]$. Otherwise, for all $\mathcal{N}$ there is $\mathcal{B}$ such that the associated machine $\mathcal{T}$ decides $\mathcal{N}$ on infinitely many input lengths. Hence, at least one part of the disjunctive statement must hold.

Proof of Lemma 6.9. Fix an input length n. For $x, y \in \{0,1\}^n$, let $G_1(y)$ be the configuration graph of $\mathcal{N}(y)$, and $G_2(x)$ be the configuration graph of $\mathcal{B}(x)$. Note that G_1 has $2^{n^{1/2+\varepsilon/2}}$ vertices and G_2 has 2^n vertices (for clarity, we ignore lower order factors in the size of configuration graphs).

The machines $\mathcal{S}, \mathcal{T}$ work as follows.

- The machine $\mathcal{S}$ attempts to compute $\mathcal{B}(x)$ for each $x \in \{0,1\}^n$ in sequence. To do so, we enumerate over $y \in \{0,1\}^n$ and attempt to run the algorithm $\mathcal{A}_2(G_1(y), G_2(x))$ of Theorem 6.7 on these inputs. If the algorithm returns a matrix $\widetilde{\mathbf{G}}$ that approximates $\mathbf{G}_2^{2^n}$, we use this matrix to determine the accepting probability of $\mathcal{B}(x)$ to within error $1/n$, and thus decide x correctly. If it returns $\perp$, we increment y and try again, and if we exhaust all choices for y, we return $\perp$. The space complexity of this algorithm is immediate by Theorem 6.7.
- The machine $\mathcal{T}$ attempts to compute $\mathcal{N}(y)$ for each $y \in \{0,1\}^n$ in sequence. To do so, we enumerate over $x \in \{0,1\}^n$ and attempt to run the algorithm $\mathcal{A}_1(G_1(y), G_2(x))$ of Theorem 6.7 on these inputs. If the algorithm returns the transitive closure of $G_1(y)$ we decide $\mathcal{N}(y)$ using this information, and otherwise increment x, and if we exhaust y return $\perp$. The time and space complexity of this algorithm is immediate by Theorem 6.7.

We first claim that one algorithm always prints the entire truth table, as the fact that such a truth table is correct follows from the above description. If for every x there is y such that $\mathcal{A}_2(G_1(y), G_2(x))$ returns a value, we print the truth table of $\mathcal{B}(x)$. Otherwise, there is some x such that $\mathcal{A}_2(G_1(y), G_2(x)) = \perp$ for every y, and by Theorem 6.7 we must have that $\mathcal{A}_1(G_1(y), G_2(x))$ returns a value for every y, and thus we print the truth table of $\mathcal{N}$. □

A very similar argument establishes Theorem 6.10:

Theorem 6.10. *At least one of the following holds:*

- $\mathsf{BPSPACE}[n] \subseteq \mathsf{NSPACE}[O(n)]$.
- $\mathsf{NSPACE}[n] \subseteq i.o.\mathsf{TISP}\left[2^{O(n)}, n^{O(1)}\right]$.

Let $\mathcal{B}$ be an arbitrary $\mathsf{BPSPACE}[n]$ machine and $\mathcal{N}$ be an arbitrary $\mathsf{NSPACE}[n]$ machine. The following lemma implies Theorem 6.10 in exactly the same way as Lemma 6.9 implies Theorem 6.8.

Lemma 6.11. *There is an* $\mathsf{NSPACE}[O(n)]$ *machine* $\mathcal{M}$ *and a* $\mathsf{TISP}[2^{O(n)}, \mathrm{poly}(n)]$ *machine* $\mathcal{T}$ *such that on every input length* n*, either* $\mathcal{M}$ *prints the truth table of* $\mathcal{B}$ *on inputs of length* n*, or* $\mathcal{T}$ *prints the truth table of* $\mathcal{N}$ *on input of length* n*.*

Proof. Fix an input length n. For $x, y \in \{0,1\}^n$, let $G_1(y)$ be the configuration graph of $\mathcal{N}(y)$, and $G_2(x)$ be the configuration graph of $\mathcal{B}(x)$. Note that G_1 and G_2 both have 2^n vertices.

The machines $\mathcal{M}, \mathcal{T}$ work as follows.

- The machine $\mathcal{M}$ attempts to compute $\mathcal{B}(x)$ for each $x \in \{0,1\}^n$ in sequence. To do so, we enumerate over $y \in \{0,1\}^n$ and attempt to run the algorithm $\mathcal{A}_2(G_1(y), G_2(x))$ of Theorem 6.6 on these inputs, making nondeterministic guesses and halting if the guess sequence is bad. If the algorithm returns a matrix $\widetilde{\mathbf{G}}$ that approximates $\mathbf{G}_2^{2^n}$, we use this matrix to determine the accepting probability of $\mathcal{B}(x)$ to error $1/n$, and thus decide x correctly. If it returns $\perp$, we increment y and try again, and if we exhaust all choices for y return $\perp$. The space complexity of this algorithm is immediate by Theorem 6.6.
- The machine $\mathcal{T}$ attempts to compute $\mathcal{N}(y)$ for each $y \in \{0,1\}^n$ in sequence. To do so, we enumerate over $x \in \{0,1\}^n$ and attempt to run the algorithm $\mathcal{A}_1(G_1(y), G_2(x))$ of Theorem 6.6 on these inputs. If the algorithm returns the reachability matrix of $G_1(y)$ we decide $\mathcal{N}(y)$ using this information, and otherwise increment x, and if we exhaust y return $\perp$. Similarly, the time and space complexity of the algorithm readily follows from Theorem 6.6.

The fact that one algorithm always prints the entire truth table follows exactly as in the proof of Lemma 6.9. □

6.3 Derandomization and Isolation From Weaker Assumptions

Our improved hardness for derandomizing space is a direct consequence of Lemma 6.2, as we do not need to construct a bootstrapping system.

Theorem 4. *There is a constant* $c > 1$ *such that the following holds. Suppose there exists a constant* $\varepsilon > 0$ *such that* $\mathsf{SPACE}[n]$ *is hard for* $\mathsf{TISP}[2^{cn}, n^c]$*-uniform circuits of size* n^c *with oracle access to* $\mathsf{SPACE}[\varepsilon n]$*.*[33] *Then,* $\mathsf{BPSPACE}[n] \subseteq \mathsf{SPACE}[O_\varepsilon(n)]$.

Proof. Let $L_{hard} \in \mathsf{SPACE}[n]$ be our hard language, and let $\mathcal{S}$ be the machine that computes it. Let $\mathcal{B}$ be an arbitrary machine computing a language in $\mathsf{BPSPACE}[n]$. For an input $x \in \{0,1\}^n$, let G_x be the configuration graph of $\mathcal{B}$, over $N_V = 2^{O(n)}$ vertices. Denote $N = N_V^{1/\varepsilon}$ – the input length that we consider for L_{hard}. We let $T_{G_x} \colon \{0,1\}^{O(\log^2 N_V) = O(n^2)} \to \{0,1\}$ be the indicator defined in Theorem 4.8, and we also use the same $t = 50 \log N_V$ defined there.

[33] The input length n to the oracle is the same length as the input to the generating algorithm (so we do not let the machine write longer oracle queries).

The Derandomization. Let $f \in \{0,1\}^{2^N}$ be the truth table of L_{hard} on inputs of length N. On input x to $\mathcal{B}$, consider the following deterministic machine $\mathcal{D}$.

- Run $\mathsf{GEN}(G_x, f)$, where GEN is the generator from Lemma 6.2.[34] We assume here that it outputs some $\vec{h}$ (and not $\perp$), as otherwise we will soon see that we get a contradiction to our hardness assumption.
- Letting $\mathbf{G}$ be the transition matrix of G_x, the machine $\mathcal{D}$ computes $\mathbf{G}_{t,\vec{h}}$, and accepts iff the corresponding entry[35] in $\mathbf{G}_{t,\vec{h}}$ is greater than $\frac{1}{2}$.

For correctness, recall that $\vec{h}$ is such that $T_{G_x}(\vec{h}) = 1$, and so

$$\|\mathbf{G}_{t,\vec{h}} - \mathbf{G}^{N_V}\| \leq N_V^{-2}.$$

In particular, if $\mathbb{E}[\mathcal{B}(x, \mathbf{U})] \geq \frac{2}{3}$ then $\mathcal{D}$ accepts, and if $\mathbb{E}[\mathcal{B}(x, \mathbf{U})] \leq \frac{1}{3}$ it rejects. For the space requirements, recall that GEN is computable in $O(\log N) = O(\frac{1}{\varepsilon}n)$ space. Computing $\mathbf{G}_{t,\vec{h}}$ takes $O(\log N_V) = O(n)$ space (see Claim 4.5), and the rest of the operations are elementary.

The Reconstruction. Now, assuming that there exists an $x \in \{0,1\}^n$ for which $\mathsf{GEN}(G_x, f) = \perp$. The reconstruction algorithm R_{f} enumerates over all x-s until it finds one. This takes $O(\text{polylog } N) = O(\text{poly}(n/\varepsilon))$ space and $\text{poly}(N_V) \cdot \text{polylog}(N) = 2^{O(n)}$ time. Then,

- R_{f} runs $\mathsf{REC}(G_x, f)$, which outputs a circuit C of size $\text{polylog}(N) = \text{poly}(n/\varepsilon)$, and a constant-size description of an oracle machine $\mathcal{M}$ that runs in space $O(\log N_V) = O(n)$, such that $C^{\mathcal{M}^G}$ computes f.
- We let R_{f} hard-wire the description of $\mathcal{M}$, $\mathcal{B}$, and x. Oracle calls to the machine $\mathcal{M}^G$ can then be simulated by a machine that runs in space $O(n)$, noticing that calls to G_x, given x and the description of $\mathcal{B}$, can be simulated in space $O(\log N_V) = O(n)$.

Overall, R_{f} runs in space $O(\text{polylog } N) = \text{poly}(n/\varepsilon)$ and time $2^{O(n)}$ and outputs a circuit of size $\text{polylog}(N) = \text{polylog}(n/\varepsilon)$ with oracle calls to a fixed language in $\mathbf{SPACE}[O(n)]$ that computes f. □

Theorem 6.12 (Theorem 6, stronger version)**.** *There is a constant $c > 1$ such that the following holds. Suppose there exists a constant $\varepsilon > 0$ such that $\mathbf{USPACE}[n] \cap \mathbf{coUSPACE}[n]$ is hard for uniform oracle circuits of size n^c with access to a fixed oracle in $\mathbf{USPACE}[\varepsilon cn] \cap \mathbf{coUSPACE}[\varepsilon cn]$, where the circuits themselves are uniformly generated by an algorithm that runs in $\mathbf{TISP}[2^{O(n)}, \text{poly}(n)]$ with oracle access to a fixed language in $\mathbf{USPACE}[O(n)] \cap \mathbf{coUSPACE}[O(n)]$.*[36] *Then, $\mathbf{NSPACE}[n] \subseteq \mathbf{USPACE}[O_\varepsilon(n)]$.*

Proof. Letting $L_0 \in \mathbf{NSPACE}[n]$ be the language decidable by a nondeterministic machine $\mathcal{N}$ that we want to decide in unambiguous space, for a fixed input $x \in \{0,1\}^n$, we let $G_x = (V, E)$ be the configuration graph of $\mathcal{N}(x)$, over $N_V = 2^{O(n)}$ vertices. In [LPT24], the indicator T_G took as input a string of length $\widetilde{O}(N_V^2)$ that encoded a weight function $E \to [\text{poly}(|V|)]$. Here, recall that T_G gets as input an $m = c_m \cdot \log^2 N_V$-bits string z, and outputs $G_{\mathsf{vMP}}(z)$ as the candidate weight function. The proof then follows the proof of [LPT24, Theorem 6.4], with the following three modifications.

[34] In Lemma 6.2 we use $N = N_V^2$, but it is easy to see that one can use any constant power. Importantly, the machine M still runs in space $O(\log N_V)$, rather than $O(\log N)$.

[35] We can assume without loss of generality that $\mathcal{B}$ has a unique accepting configuration, and then the entry is simply (s_0, s_A), for s_0 being the initial configuration, and s_A being the accepting one.

[36] Here too, the input length n to the oracle is the same length as the input to the generating algorithm.

1. In [LPT24], they use the Nisan–Wigderson generator NW to output $\widetilde{O}(N^2)$ pseudorandom bits, whereas we only need to output $m \ll N_V$ bits, which allows us to work with much weaker hardness assumption. And indeed, at the low-end regime (where the NW generator is not applicable), we use the SU somewhere-random PRG, similarly to the way we used it in Lemma 6.2.

2. In [LPT24], the reconstruction procedure computes the advantage of every D2P outcome P_i, over NW^f, and is guaranteed to find one with a large enough advantage. Here, we need to find several P_i-s, one for each list that the somewhere-PRG outputs, and moreover, the algorithm that generates $C^{\tilde{P}}$ uses randomness. Again, this is similar to what we did in Lemma 6.2 in the deterministic setting.

3. Finally, in [LPT24], they hard-wire a graph G to the circuit that computes the hard function f. We cannot afford to do that.

We begin with addressing Item 1. Set $N = N_V^{1/\varepsilon}$, and $M(N) = c_m \log^2(N^\varepsilon)$, noting that $M(N) = m$. Let

$$\mathsf{SU}^f \colon \{0,1\}^{d=O(\log N)} \times [\ell = O(\log(N)/\log(M))] \to \{0,1\}^m$$

be the SU generator from Theorem 5.1, where f is the hard truth-table given by the language $L_{hard} \in \mathbf{USPACE}[n] \cap \mathbf{coUSPACE}[n]$ which we assume is hard, on inputs of length N. Note, moreover, that $d, \ell = O(\frac{1}{\varepsilon} \cdot n)$. For the derandomization algorithm, similarly to [LPT24], we enumerate over $(y,i) \in \{0,1\}^d \times [\ell]$ and check (using the unambiguous logspace machine of [AM08] that runs in $\mathbf{USPACE}[O(\log N_V)] \cap \mathbf{coUSPACE}[O(\log N_V)]$) whether the weight function $G_{\mathsf{vMP}}(\mathsf{SU}^f(y,i))$ induces unique shortest paths in G_x. The space analysis uses the fact that the generator can be computed in space $O(\log N)$ with oracle calls to f, which also applies to the SU generator, so the derandomization runs in (unambiguous) space $O(\log N) = O(\frac{1}{\varepsilon} \cdot n)$.

As noted before, the benefit of the SU generator over NW is that its reconstruction procedure outputs an oracle circuit of size $\mathrm{poly}(m) \ll N^\varepsilon$ that computes the hard function. However, it is also a somewhere-PRG (in our space-efficient implementation), and also the algorithm that produces the circuit uses randomness, which brings us to deal with Item 2. For our D2P transformation, we instantiate Theorem 4.11 with G_x and $\delta = m^{-2}$ (which is what's needed for Item 2 of Theorem 5.1), letting $P_1, \ldots, P_b$ for $b = O(m^4) = \mathrm{poly}(n)$ be the candidate predictors, and recall that the function $(G, i, x) \to P_i(x)$ is computable in $\mathbf{USPACE}[O(n)] \cap \mathbf{coUSPACE}[O(n)]$, since $O(\log N_V) = O(n)$.

In [LPT24], they enumerate over all P_i-s until a δ-predictor is found. Here, in a very similar way (noting that this should be done in an unambiguous way), in space $O(\log N)$ we can unambiguously compute the mapping $i \to k_i$ such that for every i, P_{k_i} is a δ-predictor for $L_i = \mathsf{SU}^f(\mathbf{U}_\ell, i)$. Letting R be the reconstruction algorithm of the (somewhere-random) PRG, we give it oracle access to $P_{k_1}, \ldots, P_{k_q}$ via the $i \to k_i$ and $(G_x, k_i, x) \to P_{k_i}(x)$ transformations. Now, for any *fixed* randomness of R, call it $w \in \{0,1\}^{O(\log N)}$, all oracles are unambiguous machines so there is exactly one guess sequence where we output a circuit, which we denote by C_w. To output the correct C, via a machine which we denote R_{f}, we enumerate over all $w \in \{0,1\}^{O(n)}$, and for each one:

- R_{f} calls R to compute C_w and writes it to the work tape. This can be done in unambiguous space $O(\log N)$, and in particular can be implemented in deterministic space $O(\log N) = O(n)$ and oracle calls to a $\mathbf{USPACE}[O(n)] \cap \mathbf{coUSPACE}[O(n)]$ language.
- R_{f} enumerates over all $x_0 \in [N]$.[37]

[37] Note the difference between x_0 and x. Recall that, as in [LPT24], we go over all x's until we find one such that

- It checks whether $C_w(x_0) = L_{hard}(x)$, recalling that $L_{hard} \in \mathbf{USPACE}[n] \cap \mathbf{coUSPACE}[n]$. The evaluation $C_w(x_0)$ can be done in $\mathrm{poly}(n)$ time using our oracle access. If the computation of $L_{hard}(x)$ returned $\neg C_w(x_0)$, abort and proceed to the next w.
- When we find a w that succeeds for all x_0's, set $C = C_w$.

We then have that R_{f} runs in (deterministic) time $2^{O(n)}$, uses $\mathrm{poly}(n)$ space, and has oracle access to a (fixed) language in $\mathbf{USPACE}[O(n)] \cap \mathbf{coUSPACE}[O(n)]$. Note that, as in [LPT24], we implement the *predictor calls* in our circuit C using the

$$\mathbf{USPACE}[O(\log N_V)] \cap \mathbf{coUSPACE}[O(\log N_V)] = \mathbf{USPACE}[O(\varepsilon \log N)] \cap \mathbf{coUSPACE}[O(\varepsilon \log N)]$$

oracle gates (while our repeated invocations of R is allowed to run in larger unambiguous space, namely $O(\log N)$).

All that is left is to address Item 3. In [LPT24], the graph G_x (together with a D2P index i) was hard-wired by R in order to compute the mapping $x \to P_i(x)$. For us, hard-wiring the graph is too costly. Instead, we hard-wire the (assumed towards a contradiction) $x \in \{0,1\}^n$, and the (constant-size) encoding of the TM for $L_0 \in \mathbf{NSPACE}[n]$. We then use the fact that the transformation $(x, L_0) \to G_x$ can be computed in deterministic space $O(\log N_V)$, and simply utilize the circuit's oracle gates to perform the transformation. Overall, R_{f} runs in

$$\mathbf{TISP}[2^{O(n)}, \mathrm{poly}(n)]^{\mathbf{USPACE}[O(n)] \cap \mathbf{coUSPACE}[O(n)]}$$

and prints a circuit of size $\mathrm{poly}(M) = \mathrm{poly}(\log^2 N_V) = \mathrm{poly}(n)$, and with oracle gates to

$$\mathbf{USPACE}[O(\log N_V)] \cap \mathbf{coUSPACE}[O(\log N_V)].$$

This concludes the modifications over [LPT24]. □

6.4 Minimal-Memory Derandomization

We now prove our results that deduce derandomization with minimal memory overhead from hardness of deterministic compression. The main new technical tool that we rely on is the D2P transformation for the FK generator composed with an AOBP, which was presented in Section 4.3. Specifically, we will prove the following result.

Assumption 6.13. The assumption is parametrized by constants $C > 1$ and $\varepsilon, \delta > 0$. There is a function $f\colon \{0,1\}^\star \to \{0,1\}^\star$ that maps n bits to n^2 bits and is computable in space $(C + 1 + \varepsilon + \delta) \cdot \log(n)$ that satisfies the following. For every deterministic algorithm R that runs in space n^ε and time $n^{O(C)}$, there are at most finitely many $x \in \{0,1\}^\star$ for which $R(x)$ prints a Turing machine M of description size $O(|x|)$ that runs in space $(C + 1 + \varepsilon) \cdot \log(|x|)$ such that $M(x) = f(x)$.

Theorem 6.14. *Suppose that Assumption 6.13 is true for some C, ε, δ. Then, for $S(n) = C \cdot \log(n)$ we have that*

$$\mathbf{BPSPACE}[S] \subseteq \mathbf{SPACE}[2S + (1 + \delta + (c/\varepsilon)) \cdot \log(n)],$$

where $c > 1$ is a universal constant.

SU^f does not induce unique shortest paths in G_x, and that's the graph we work with. We enumerate over the x_0's once we fixed an x.

The rest of this section contains the proof of Theorem 6.14. The statement in the intro, Theorem 5, follows by a standard padding argument, since the assumption implies that Assumption 6.13 holds for an arbitrarily large constant C, $\varepsilon = 0.01$, and $\delta = 3$. Going forward, we will need two technical tools. The first is a memory-efficient implementation of randomized space-bounded algorithms, from [DT23]:

Lemma 6.15 ([DT23], see also [DPT24, Lemma 7.1])**.** *For any randomized space-S machine M there is a randomized oracle machine $\bar{M}$ that works as follows. The machine $\bar{M}$ runs in space $S + O(\log S)$, and whenever $\bar{M}$ queries a random bit while in configuration τ, it queries the random oracle at position τ. Moreover, for every $x \in \{0,1\}^n$ it holds that*

$$\Pr_{r}[M(x,r) = 1] = \Pr_{r'}[\bar{M}^{r'}(x) = 1],$$

and the computation of $\bar{M}$ on x as a function of r' can be simulated by an AOBP of length and width 2^S.

The second technical component that we need, from [DPT24], is an implementation of the NW PRG [NW94] whose reconstruction is a logspace-uniform $\mathbf{TC}^0$ circuit (this uses a code for the hardness amplification step with highly efficient decoding).

Theorem 6.16 (NW PRG with a deterministic $\mathbf{TC}^0$ reconstruction; see [DPT24, Theorem 7.4])**.** *There is a universal constant $c_{\mathsf{NW}} > 1$ such that for every sufficiently small constant $\varepsilon_{\mathsf{NW}} > 0$ the following holds. There is an algorithm NW computing*

$$\mathsf{NW}^f \colon \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\cdot\log(N)} \to \{0,1\}^M$$

such that for any $f \in \{0,1\}^N$ and for $M = N^{\varepsilon_{\mathsf{NW}}}$ the following holds.

1. ***Efficiency.*** *The mapping $(s,i) \mapsto \mathsf{NW}^f(s)_i$ is computable in space $(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log(N)$.*
2. ***Reconstruction.*** *There is a deterministic space-$O(\log N)$ algorithm R that, given oracle access to f and oracle access to a $(1/M^2)$-next-bit-predictor P for NW^f, prints a constant-depth oracle circuit C of size $M^{c_{\mathsf{NW}}}$ that has majority gates, makes non-adaptive queries, and satisfies $C^P(x) = f_x$ for all $x \in [N]$.*

Let us now turn to the proof of Theorem 6.14. Let $L \in \mathbf{BPSPACE}[S]$, and let M be a randomized space-S machine deciding L. Let $\bar{M}$ be as in Lemma 6.15. Let $N = n^2$ and $\ell = \log(N)$, and $\varepsilon_{\mathsf{NW}} = \varepsilon/2c_{\mathsf{NW}}$, where c_{NW} is the universal constant from Theorem 6.16.

The deterministic machine A that decides L is defined as follows. On input x let $f = f(x) \in \{0,1\}^N$, and let

$$\mathsf{NW}^f \colon \{0,1\}^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\cdot\ell} \to \{0,1\}^{n^{\varepsilon_{\mathsf{NW}}}}$$

be the NW PRG from Theorem 6.16. For $C' = C + c'$ where $c' > 1$ is a sufficiently large universal constant, also let

$$\mathsf{FK} \colon \{0,1\}^{n^{\varepsilon_{\mathsf{NW}}}} \to \{0,1\}^{n^C}$$

be the generator from Theorem 4.14 with output length $N = n^C$ and parameter $\varepsilon_{\mathsf{NW}}/C'$. Denoting $\bar{N} = 2^{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}})\cdot\ell} = N^{c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}}$, the algorithm A is defined as

$$A(x) = \mathsf{MAJ}_{s\in[\bar{N}]}\left\{\bar{M}^{\mathsf{FK}(\mathsf{NW}^f(s))}(x)\right\}.$$

Space complexity. We implement A using standard space-efficient composition. Since some steps will require extra care, let us spell out parts of the construction.

- Enumerate over $s \in [\bar{N}]$ while keeping a counter for the outcomes of $\bar{M}$.
- For a fixed s, run $\bar{M}$ on input x.
- Whenever $\bar{M}$ queries its oracle at location $j \in [n^C]$, simulate FK on input $\mathsf{NW}^f(s)$.
- Whenever FK queries its virtual input, compute f and NW using standard space-bounded composition (i.e., simulate NW on input s and answer its queries to f by computing each bit of f on-the-fly).

The first key point to remember is that any query j that $\bar{M}$ makes to its oracle is the configuration of $\bar{M}$. In other words, $\bar{M}$ does not write queries using additional workspace, and to answer its query we simply compute the oracle on the configuration of $\bar{M}$ that is written on the worktape. The second key point to remember is that FK runs in space $O(\varepsilon_{\mathsf{NW}}/C) \cdot \log(N)$ with catalytic access to the query location j. Thus, whenever $\bar{M}$ makes a query $j \in [n^C]$ to the oracle, FK will change the configuration j of $\bar{M}$, but will return it to its original state after its computation is over (and when $\bar{M}$ receives the oracle answer and is ready to resume its execution).

Hence, on any input x, the algorithm A can be implemented in space

$$\underbrace{2\log(\bar{N})}_{\text{enumerating } s \text{ and outcomes of } \bar{M}} + \underbrace{c_0 \cdot (\varepsilon_{\mathsf{NW}}/C') \cdot \log(N)}_{\mathsf{FK}} + \underbrace{S + O(\log S)}_{\bar{M}} + \underbrace{(c_{\mathsf{NW}}/\varepsilon_{\mathsf{NW}}) \cdot \log(N)}_{\mathsf{NW}}$$

$$+ \underbrace{\frac{C+1+\varepsilon+\delta}{2} \cdot \log(N)}_{f} + \underbrace{c_0 \cdot (\log N)}_{\text{compositional overheads}} \leq (2C + 1 + \varepsilon + \delta + (c/\varepsilon)) \cdot \log(n)$$

for some universal constants $c_0, c > 1$. Recalling that $S = C \cdot \log(n)$, the space complexity of this algorithm is at most

$$2S + (1 + \delta + 2c/\varepsilon) \cdot \log(n).$$

Analysis. Let x such that $A(x) \neq L(x)$. We will show that in this case, $f(x)$ can be compressed in space n^ε and time $\mathrm{poly}(n^C)$ to a Turing machine of description length $n + n^\varepsilon$ that uses space at most $(1+\varepsilon) \cdot C \cdot \log(n)$. By our assumption, this cannot be done for more than finitely many inputs.

For any input x such that $A(x) \neq L(x)$ we have that $D_x(r) = \bar{M}^r(x)$ is a $(1/10)$-distinguisher for the mapping $s \mapsto \mathsf{FK}(\mathsf{NW}^f(s))$. Since D_x is an AOBP of size $2^S = n^C$, the generator FK fools it up to error $1/n^{2C}$. Thus,

$$\left| \Pr_{r \leftarrow \mathbf{U}}[D_x(r) = 1] - \Pr_{s \leftarrow \mathbf{U}}[D_x(\mathsf{FK}(\mathsf{NW}^f(s))) = 1] \right| > 1/6,$$
$$\left| \Pr_{r \leftarrow \mathbf{U}}[D_x(r) = 1] - \Pr_{w \leftarrow \mathbf{U}}[D_x(\mathsf{FK}(w)) = 1] \right| \leq 1/n^{2C},$$

which implies that, denoting $D'_x(w) = D_x(\mathsf{FK}(w))$, we have

$$\left| \Pr_{w \leftarrow \mathbf{U}}[D'_x(w) = 1] - \Pr_{s \leftarrow \mathbf{U}}[D'_x(\mathsf{NW}^f(s)) = 1] \right| > 1/10.$$

Given x, to compress $f(x)$ we run the reconstruction algorithm R from Theorem 6.16. Recall that R needs query access to f and to a $(1/N^{2\varepsilon_{\mathsf{NW}}})$-next-bit-predictor P; we answer queries to f by computing $f(x)$, and we answer queries to a predictor P using the D2P algorithm from Theorem 4.14 (and simulating the output of the D2P on the query; see more on that below). In turn, the D2P algorithm needs query access to NW^f, which we provide by simulating NW and computing $f(x)$ as needed. The reconstruction then prints a $\mathbf{TC}^0$ circuit B of size $N^{c_{\mathsf{NW}} \cdot \varepsilon_{\mathsf{NW}}}$ such that $\mathsf{tt}(B^P) = f(x)$, where P is the predictor given by the D2P algorithm.

Note that the D2P algorithm runs in space $O(n^{\varepsilon_{\mathsf{NW}}})$ and time $\mathrm{poly}(n^C)$, and thus its space and time complexity dominates the entire compression algorithm. In particular, whenever R queries P, since we are using space $O(n^{\varepsilon_{\mathsf{NW}}})$ anyway to evaluate the D2P algorithm, we can store the entire output of the D2P algorithm, which is a predictor P, and evaluate P at the given query location. Thus, overall, the compression algorithm runs in space $O(n^{\varepsilon_{\mathsf{NW}}}) < n^{\varepsilon}$ and time $\mathrm{poly}(n^C)$.

Turning to correctness, note that in both cases spelled out in Theorem 4.14, the D2P algorithm can print a description of length at most $n + n^{\varepsilon/3}$ of an $n^{-c' \cdot (\varepsilon_{\mathsf{NW}}/C')}$-next-bit-predictor, where $c' > 1$ is a universal constant from Theorem 4.14 (i.e., either the machine L whose size is logarithmic, or a description of D_x along with the suffix z and (b, i), which can be specified using at most $n + n^{\varepsilon/3}$ bits[38]). Relying on a sufficiently large choice of $C' > C + c'$, this is a $1/N^{2\varepsilon_{\mathsf{NW}}}$-next-bit-predictor

The total length needed to describe B and the predictor P is thus at most $n + n^{\varepsilon/2}$. The compression algorithm prints a description of a Turing machine $\mathcal{M}$ that has B and P hard-wired, and given input $j \in [N]$ it evaluates B on j using the standard space-efficient DFS simulation, while answering queries using P. The total description length is at most $n + n^{\varepsilon}$, and the space complexity of $\mathcal{M}$ is $n^{O(\varepsilon_{\mathsf{NW}})} + \mathsf{Space}(P)$, where in both cases of Theorem 4.14 we have $\mathsf{Space}(P) \leq (C + 1 + O(\varepsilon_{\mathsf{NW}})) \cdot C \cdot \log(n) < (C + 1 + \varepsilon) \cdot \log(n)$.

Acknowledgments

E.P. thanks Joshua Cook, Oded Goldreich, and Dana Moshkovitz for helpful conversations. This work benefited from the participation of Dean Doron and of Roei Tell in Dagstuhl Seminar 24381, where part of the work on this project was conducted.

References

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

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

[AM08] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

[38] Specifically, to specify D_x we just need to specify x and the machines M and FK, where the latter two have constant-sized descriptions.

[ATWZ00] Roy Armoni, Amnon Ta-Shma, Avi Wigderson, and Shiyu Zhou. An $O(\log(n)^{4/3})$ space algorithm for $(s,\ t)$ connectivity in undirected graphs. *J. ACM*, 47(2):294–311, 2000.

[BBRS98] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proc. 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 276–287, 1994.

[CCvM06] Jin-yi Cai, Venkatesan T. Chakaravarthy, and Dieter van Melkebeek. Time-space tradeoff in derandomizing probabilistic logspace. *Theory Comput. Syst.*, 39(1):189–208, 2006.

[CH22] Kuan Cheng and William M. Hoza. Hitting sets give two-sided derandomization of small space. *Theory Comput.*, 18:1–32, 2022.

[CIKK15] Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Tighter connections between derandomization and circuit lower bounds. In *Proc. 19th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 645–658, 2015.

[CLO+23] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. *arXiv preprint arXiv:2305.15140*, 2023.

[CLTW23] Lijie Chen, Xin Lyu, Avishay Tal, and Hongxun Wu. New PRGs for unbounded-width/adaptive-order read-once branching programs. In *Proc. 50 International Colloquium on Automata, Languages and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 39:1–39:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[CR80] Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980.

[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.

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

[CT23] 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.

[DMOZ22] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *Journal of the ACM*, 69(6):1–55, 2022.

[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.

[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.

[EPA99] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6):2257–2284, 1999.

[FK18] Michael A. Forbes and Zander Kelley. Pseudorandom generators for read-once branching programs, in any order. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 946–955, 2018.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

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

[GRZ23] Uma Girish, Ran Raz, and Wei Zhan. Is untrusted randomness helpful? In *Proc. 14 Conference on Innovations in Theoretical Computer Science (ITCS)*, volume 251 of *LIPIcs*, pages 56:1–56:18, 2023.

[GSV18] Aryeh Grinberg, Ronen Shaltiel, and Emanuele Viola. Indistinguishability by adaptive procedures with advice, and lower bounds on hardness amplification proofs. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 956–966, 2018.

[Guo13] Zeyu Guo. Randomness-efficient curve samplers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 575–590. Springer, 2013.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *Journal of the ACM*, 56(4):Art. 20, 34, 2009.

[GW96] Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996.

[Hir23] Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. *SIAM Journal on Computing*, 52(6):FOCS18–349–FOCS18–382, 2023.

[Hoz21] William M. Hoza. Better pseudodistributions and derandomization for space-bounded computation. In *Proceedings of the 25th International Conference on Randomization and Computation (RANDOM)*, pages 28:1–28:23, 2021.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

[ISW06] Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. Reducing the seed length in the Nisan-Wigderson generator. *Combinatorica*, 26(6):647–681, 2006.

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

[Jof74] Anatole Joffe. On a set of almost deterministic k-independent random variables. *the Annals of Probability*, 2(1):161–162, 1974.

[KRC00] Valentine Kabanets, Charles Rackoff, and Stephen A. Cook. Efficiently approximable real-valued functions. *Electron. Colloquium Comput. Complex.*, TR00-034, 2000.

[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.

[LPT24] Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness, 2024.

[LZPC05] Pinyan Lu, Jialin Zhang, Chung Keung Poon, and Jin-yi Cai. Simulating undirected st-connectivity algorithms on uniform JAGs and NNJAGs. In *Proceedings of 16th International Symposium on Algorithms and Computation (ISAAC)*, volume 3827 of *Lecture Notes in Computer Science*, pages 767–776. Springer, 2005.

[Nis91] Noam Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis94] Noam Nisan. $\mathsf{RL} \subseteq \mathsf{SC}$. *Computational Complexity*, 4:1–11, 1994.

[NSW92] Noam Nisan, Endre Szemerédi, and Avi Wigderson. Undirected connectivity in $o(\log^{1.5} n)$ space. In *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 24–29, 1992.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[Poo93] Chung Keung Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *34th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 218–227. IEEE Computer Society, 1993.

[PP23] Aaron (Louie) Putterman and Edward Pyne. Near-optimal derandomization of medium-width branching programs. In *Proc. 55 Annual ACM Symposium on Theory of Computing (STOC)*, pages 23–34, 2023.

[PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, 2023.

[Pyn24] Edward Pyne. Derandomizing logspace with a small shared hard drive. In *Proc. 39th Annual IEEE Conference on Computational Complexity (CCC)*, pages 4:1–4:20, 2024.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):17:1–17:24, 2008.

[RSW06] Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. *SIAM Journal on Computing*, 35(5):1185–1209, 2006.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[STV01] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52(2):172–216, 2005.

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complex.*, 13(1):180–193, 1997.

[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.

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

[SZ99] Michael E. Saks and Shiyu Zhou. $\mathbf{BP_HSPACE}[S] \subseteq \mathbf{DSPACE}[S^{3/2}]$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[TSUZ07] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[TU06] Amnon Ta-Shma and Christopher Umans. Better lossless condensers through derandomized curve samplers. In *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 177–186, 2006.

[TZS06] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed-Muller codes. *Journal of Computer and System Sciences*, 72(5):786–812, 2006.

[vMP19] Dieter van Melkebeek and Gautam Prakriya. Derandomizing isolation in space-bounded settings. *SIAM J. Comput.*, 48(3):979–1021, 2019.

[VV86] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[Wig92] Avi Wigderson. The complexity of graph connectivity. In *Mathematical Foundations of Computer Science (MFCS)*, volume 629 of *Lecture Notes in Computer Science*, pages 112–132. Springer, 1992.

[Yao82] Andrew C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.