Catalytic Tree Evaluation from Matching Vectors

Alexandra Henzinger* MIT, Stanford
Edward Pyne† MIT
Seyoon Ragavan‡ MIT

November 8, 2026

Abstract

We give new algorithms for tree evaluation (S. Cook et al. TOCT 2012) in the catalytic-computing model (Buhrman et al. STOC 2014). Two existing approaches aim to solve tree evaluation (TreeEval) in low space: on the one hand, J. Cook and Mertz (STOC 2024) give an algorithm for TreeEval running in super-logarithmic space $O(\log n \log\log n)$ and super-polynomial time $n^{O(\log\log n)}$. On the other hand, an earlier algorithm of J. Cook and Mertz (STOC 2020), run on a catalytic tape, solves TreeEval in logarithmic $O(\log n)$ free space and polynomial time, but with polynomial $O(\sqrt{n})$ catalytic space.

We show that the latter result can be improved. We give a catalytic algorithm for TreeEval with logarithmic $O(\log n)$ free space, polynomial runtime, and subpolynomial $2^{O(\log^{\varepsilon} n)}$ catalytic space (for any $\varepsilon > 0$). Our result also implies an improved simulation of time by *catalytic* space, via the reduction of Williams (STOC 2025).

Our TreeEval algorithm comes from a framework, inspired by private information retrieval, that generically reduces tree evaluation to two combinatorial objects: matching-vector families and *decoders* (a new object we define that closely resembles decoding polynomials). These ingredients' parameters directly govern an algorithm's resources: the vectors' dimension determines its catalytic space, and the decoder's sparsity determines its free space and runtime. In addition to enabling our new algorithm, this framework explains existing tradeoffs, for example why the Cook–Mertz algorithm needs super-logarithmic space but no catalytic tape. It also lets better matching-vector families or decoders plug in directly to give better algorithms.

*Email: amh@cs.stanford.edu. Supported by the NSF Graduate Research Fellowship under Grant No. 2141064, a MIT EECS Great Educators fellowship, gifts from Apple, Google, and Meta, and NSF CNS-2054869.

†Email: epyne@mit.edu. Supported by the NSF Graduate Research Fellowship.

‡Email: sragavan@mit.edu. Supported by Amazon Research Awards and NSF CNS-2534400, and in part by Jane Street. SR's access to ChatGPT Pro was supported by the UK AISI alignment project and the Simons Institute for the Theory of Computing, and his access to Claude Max was supported by Jane Street.

Contents

1 Introduction **1**
- 1.1 Related Work . . . 3
- 1.2 Open Questions . . . 4

2 Technical Overview **4**
- 2.1 Tree Evaluation and the Cook–Mertz Algorithm . . . 4
- 2.2 This Work: A Connection between TreeEval and Private Information Retrieval . . . 7
- 2.3 Adapting Matching-Vector PIR to TreeEval . . . 9

3 Preliminaries **14**
- 3.1 The Catalytic-Space Model . . . 14
- 3.2 Existing Algorithms for Tree Evaluation . . . 15
- 3.3 Matching Vector Families . . . 16
- 3.4 A-Decoders . . . 18

4 Main Result: Catalytic Tree Evaluation from Matching Vectors **21**
- 4.1 Recursive Step: One Level of Tree Evaluation . . . 22
- 4.2 Proof of Theorem 4.4 . . . 23
- 4.3 Proof of Theorem 4.1 . . . 25
- 4.4 Instantiations . . . 26
- 4.5 Higher Arity and Blocking . . . 28

5 Application: New Time-Space-Catalytic Space Tradeoffs **30**
- 5.1 Composing with Space-Bounded Reductions . . . 30
- 5.2 The Reduction of Williams . . . 30
- 5.3 Circuit Evaluation . . . 31

6 Statement on AI Use **32**

7 Acknowledgements **32**

A Impossibility of Decoding Polynomials in Characteristic m **36**

B Verifying the Uniformity of the Matching-Vector Family **36**

C Proof of Corollary 3.11 **39**

1 Introduction

Space, unlike time, is a reusable resource. *Catalytic computing* studies the task of computing given a large amount of full memory, which may be modified but must appear unchanged at the end of the computation. A series of breakthrough results, initiated by Buhrman, Cleve, Koucký, Loff, and Speelman [BCK+14], show that access to such full, *catalytic* space enables using less *free* space (ordinary space that can be overwritten and erased) to solve problems than otherwise known. For example, Buhrman et al. [BCK+14] evaluate any uniform TC^1 circuit with $O(\log n)$ free space and $\mathsf{poly}(n)$ catalytic space, while this task—which contains a complete problem for non-deterministic logspace (NL)—is not known to be solvable with $o(\log^2 n)$ free space alone.

The power of catalytic space remains wide open. For instance, letting catalytic logspace (CL) be the set of problems solvable in $O(\log n)$ free space and $\mathsf{poly}(n)$ catalytic space, it would be consistent with current knowledge both that all polynomial-time computations can be solved in CL (i.e., $\mathsf{P} \subseteq \mathsf{CL}$) or that CL is no stronger than the class of computations done in $O(\log^2 n)$ parallel time (i.e., $\mathsf{CL} \subseteq \mathsf{NC}^2$). One critical question towards understanding this power is: how small can catalytic space be for it to be useful?

In this paper, we draw a new connection between catalytic computing and cryptography. Building on this connection, we obtain a new catalytic algorithm for the tree evaluation problem, abbreviated $\mathsf{TreeEval}$ [CMW+12]. $\mathsf{TreeEval}$ has become a central protagonist in complexity theory for its roles in

- aiming to separate polynomial time $\mathsf{P} = \mathsf{Time}(\mathsf{poly}(n))$ and logarithmic space $\mathsf{L} = \mathsf{Space}(O(\log n))$ [CM24] and
- giving new tradeoffs between space and time, namely that $\mathsf{Time}(T(n)) \subseteq \mathsf{Space}(\sqrt{T(n)\log T(n)})$ [Wil25].

This work shows that *logarithmic* free space and *subpolynomial* catalytic space suffice to solve $\mathsf{TreeEval}$:

Theorem 1.1. *For every $\epsilon > 0$, $\mathsf{TreeEval}$ can be solved in $O(\log n)$ free space, $2^{O(\log^\epsilon n)}$ catalytic space, and $\mathsf{poly}(n)$ runtime. (See Corollary 4.7 for a parameterized statement.)*

By contrast, the one-hot algorithm of Cook and Mertz [CM20, Theorem 1], run on a catalytic tape, solves $\mathsf{TreeEval}$ with superpolynomially more catalytic space: it uses $O(\log n)$ free space, $O(\sqrt{n})$ catalytic space, and polynomial runtime.[1] At the same time, a later algorithm of Cook and Mertz [CM24] solves $\mathsf{TreeEval}$ with no catalytic space, but with more free space and superpolynomially larger runtime than Theorem 1.1: it requires $O(\log n \cdot \log\log n)$ space and $n^{O(\log\log n)}$ time.

Our algorithm also admits smooth tradeoffs that shrink the catalytic space at the expense of increasing the free space and the runtime. For example, it implies the following new result:

Theorem 1.2. $\mathsf{TreeEval}$ *can be solved in $O(\log n\sqrt{\log\log n}\log\log\log n)$ free space, $\exp(\exp(O(\sqrt{\log\log n})))$ catalytic space, and $n^{O(\sqrt{\log\log n})}$ time. (See Corollary 4.8 for a parameterized statement.)*

Two direct implications follow from our new catalytic algorithm for tree evaluation:

1. New time-space tradeoffs in the catalytic-space model. Following the seminal reduction of Williams [Wil25], our result implies the following corollary:

Corollary 1.3. *Let $T = T(n) \geq n$. Every language decided by a multitape Turing machine in time T can be decided in $O(\sqrt{T})$ free space, $2^{O(T^\epsilon)}$ catalytic space, and $2^{O(\sqrt{T})}$ time, for every $\epsilon > 0$. (See Corollary 5.2.)*

[1] Evaluating a circuit for $\mathsf{TreeEval}$ with the catalytic algorithm of Buhrman et al. [BCK+14] gives a similar result, but with $\mathsf{poly}(n)$ catalytic space.

Relative to Williams' result, this shrinks the free space by a factor of $O(\sqrt{\log T})$ and shrinks the runtime by superpolynomial factors—his simulation runs in time $2^{O(\sqrt{T \log T})}$, against our $2^{O(\sqrt{T})}$—at the expense of introducing a large catalytic tape. Separately, combining the reduction of Williams with the one-hot algorithm of [CM20] gives that $\mathsf{Time}(T)$ can be decided with $O(\sqrt{T})$ free space and $2^{O(\sqrt{T})}$ catalytic space. Compared to this result, we shrink the catalytic tape by a superpolynomial factor. We state analogous tradeoffs for catalytic circuit evaluation.

2. A modular framework for tree evaluation. Our algorithm is an instance of a general framework that reduces tree evaluation to two combinatorial objects: matching-vector families and *decoders*, a new object closely resembling the decoding polynomials used for matching-vector codes. The framework generalizes the breakthrough algorithm of Cook and Mertz [CM24], and it lets us apply tools from coding theory and cryptography to questions in catalytic computing.

In particular, the core machinery driving our new catalytic TreeEval algorithm is a family of *matching vectors* [Gro00]; informally, this is a collection of vectors whose inner products over composite moduli fall in a restricted set. Matching vectors give rise to the best known locally-decodable codes in the low query-complexity regime [DGY11] and to the most communication-efficient information-theoretic private-information-retrieval (PIR) schemes in the few-server regime [DG16, GKS25]. We observe that an algorithm at the heart of Cook-Mertz's TreeEval mirrors Reed-Muller PIR; by extending this connection to other PIR schemes—including Efremenko's matching-vector protocols—we obtain new tradeoffs among time, space, and catalytic space for tree evaluation.

In this framework, the parameters of the two ingredients directly determine the resources of the resulting algorithm. Each node of the tree is evaluated with a number of recursive calls proportional to the decoder's sparsity, and the recursion stores only a term index and a few residues modulo m per level, so the sparsity (together with the modulus m) determines the free space and the running time. The algorithm's registers hold vectors from the matching-vector family, so their dimension determines the catalytic space. This refines the encoding-based view of Cook and Mertz [CM21], in which an algorithm's cost is governed by the length of an encoding of values as vectors and by the degree of the node functions under that encoding; in our framework, the matching-vector dimension and the decoder sparsity play these two roles. Table 1 places three algorithms in this framework.

Viewed this way, the two earlier algorithms of Cook and Mertz sit at opposite corners of a tradeoff. The one-hot algorithm of [CM20] pairs a decoder of constant sparsity with vectors of polynomial dimension, so it needs polynomial catalytic space. The algorithm of [CM24] uses vectors of low dimension, so it needs no catalytic tape, but its decoder has polylogarithmic sparsity over a polylogarithmic prime modulus, which costs it super-logarithmic space and super-polynomial time. Our algorithm uses matching vectors over a constant composite modulus, which give constant sparsity and subpolynomial dimension at the same time. The same framework also recovers the Stoeckl–Goldreich improvement [Sto23, Gol24] of the Cook–Mertz algorithm, by grouping levels of the tree into nodes of higher fan-in and decoding sums of more than two scores (Section 4.5).

Our result is the first use of such techniques in catalytic computing. We view this connection, and the framework it yields, as the main contribution of the paper, and we are optimistic for further applications and links between information-theoretic cryptography, coding theory, and catalytic computing. Because the framework is modular, better ingredients translate directly into better algorithms. For example, there is a large gap between known upper and lower bounds for matching-vector families, and shorter families would directly improve our algorithm—even up to the point of speeding up Cook-Mertz to polynomial time, with no loss in space. In the terms of Table 1, such a family would keep our constant modulus and sparsity

	Shared modulus	Matching-vector dimension	Decoder sparsity	Free space	Catalytic space	Time
[CM20, Thm. 1]	$O(1)$	$O(\sqrt{n})$	$O(1)$	$O(\log n)$	$O(\sqrt{n})$	$\mathsf{poly}(n)$
[CM24]	prime, $\mathsf{polylog}(n)$	$O(\frac{\log n}{\log\log n})$	$\mathsf{polylog}(n)$	$O(\log n \log\log n)$	none	$n^{O(\log\log n)}$
This work	composite, $O(1)$	$2^{O(\log^{\varepsilon} n)}$	$O(1)$	$O(\log n)$	$2^{O(\log^{\varepsilon} n)}$	$\mathsf{poly}(n)$

Table 1: Three tree-evaluation algorithms viewed through our framework, on inputs of length n. The row for [CM20] reads its one-hot algorithm in our terms (Section 2.3).

while bringing the dimension down to $O(\log n \log\log n)$, small enough to hold the registers in free space.

Theorem 1.4 (Informal)**.** *Suppose that (sufficiently uniform) matching-vector families over a constant composite modulus exist, with dimension not ruled out by known lower bounds. Then,* TreeEval *can be solved in $O(\log n \log\log n)$ ordinary space and polynomial time, with no catalytic tape at all. (See Corollary 4.10.)*

The same accounting also exposes this framework's limits as a route to putting TreeEval in logspace. For a constant modulus, the lower bound of Bhowmick, Dvir, and Lovett [BDL14], which the work of Gowers, Green, Manners, and Tao [GGMT24] makes unconditional, shows that the dimension assumed in Theorem 1.4 is optimal up to constant factors; so within this framework, $O(\log n \log\log n)$ is the best ordinary-space bound attainable with a constant modulus.

1.1 Related Work

Algorithms for tree evaluation. There has been extensive work on algorithms [Bar89, BC92, CMW+12, CM20, CM21, CM22, CM24] and lower bounds [CMW+12, Liu13, EMP18, IN19] for tree evaluation, as well as on evaluating classes of circuits in the catalytic model [BCK+14, AM25, CP25]. Though prior algorithms for tree evaluation use the catalytic model as an intermediate tool to save space, we give the first *catalytic* algorithm for tree evaluation with subpolynomial catalytic space, showing that subpolynomial catalytic space suffices to use only logarithmic free space. Previously, the best catalytic algorithm for tree evaluation was the one-hot algorithm of [CM20, Theorem 1], run on a catalytic tape, which uses $O(\sqrt{n})$ catalytic space (the algorithm implicit in the work of [BCK+14] uses $\mathsf{poly}(n)$ catalytic space).

Time-space tradeoffs. The breakthrough result of Williams [Wil25] improved on the 50 year old result of Hopcroft, Paul, and Valiant [HPV77] that $\mathsf{Time}(t) \subseteq \mathsf{Space}(O(t/\log t))$. To the best of our knowledge, the relationship between time (and space) and *catalytic* space remains wide open, and we are the first to give results in this area. It would even be consistent with current knowledge that $\mathsf{P} \subseteq \mathsf{CL}$, meaning that any size-$n$ circuit could be evaluated in $O(\log n)$ free space and $\mathsf{poly}(n)$ catalytic space.

Concurrently, a manuscript released by OpenAI [Ope26] reports a simulation of time-T deterministic one-tape computations in $O(T^{2/5} \cdot \mathsf{polylog}(T))$ free space and no catalytic space, improving on the prior one-tape simulation in $O(\sqrt{T})$ free space due to Hopcroft and Ullman [HU68]. In particular, this implies that RAM-time-T computation can be simulated in space $O(T^{4/5} \cdot \mathsf{polylog}(T))$, giving the first space-bounded simulation of RAM computation with space polynomially sublinear in T. Compared with the reduction of [Wil25], this result does not support a general multi-tape Turing machine, does not proceed via black-box reduction to TreeEval, and does not directly give a smaller catalytic-space bound or a faster algorithm for TreeEval.

Matching vectors. Yekhanin [Yek08] first used matching vectors to obtain new families of locally-decodable codes, starting a cascade of subsequent works [KY09, Efr09, Efr12, IS10, DGY11, Yek12, CFL+13]. Since then, matching-vector codes have found applications in low-communication PIR [DG16, GKS25, GR26] and other cryptographic protocols like conditional disclosure of secrets and linear secret sharing [LBA25]. To date, the best known families of matching vectors are due to Grolmusz [Gro00], building on work by Barrington, Beigel, and Rudich [BBR94].

1.2 Open Questions

Our work leaves open a number of concrete questions:

- *The space complexity of* TreeEval*:* is TreeEval indeed in classical logspace (L)? Separately, is it in simultaneous polynomial time and $\log^c n$ space for some $c < 2$?

  Building on this work, one approach to showing TreeEval $\in$ L would be to design a procedure for materializing matching vectors "on the fly" from a much more succinct representation stored in catalytic space, which could bring the space usage of our algorithm down, perhaps all the way to $O(\log n)$. A second approach would be to design shorter matching-vector families; for constant moduli, however, this route stops at $O(\log n \log\log n)$ space, as discussed after Theorem 1.4.

- Can our techniques be combined with the result of OpenAI [Ope26] to obtain a simulation of one-tape time T in $O(T^{2/5})$ free space and small catalytic space? This question is particularly interesting because it would imply a simulation of RAM-time T in free space $T^{4/5}$ and small catalytic space, since RAM machines can be simulated in the one-tape model with a quadratic slowdown. While the OpenAI result is not a direct reduction to TreeEval, it is possible that similar techniques could be applicable there.

A more open-ended direction is whether known lower bounds on families of PIR schemes or on locally-decodable codes translate to lower bounds on TreeEval. For example, prior works [BIM00] show that, in PIR, the servers must inherently run in $\Omega(n)$ time to answer PIR queries. It remains open to prove an analogous bound on the runtime of TreeEval—or, alternatively, to use preprocessing techniques from the PIR literature for circumventing this bound to speed up TreeEval [HR26].

Finally, we leave open whether it is possible to port other techniques from information-theoretic cryptography, and multi-party computation in particular, to obtain improvements in catalytic computing. Taking a step back, our methods leverage the following qualitative view: both catalytic computing and cryptographic protocols work by operating on *masked* values. In particular, catalytic computation requires intermediate results to be written onto a catalytic tape, meaning these results are stored and accessed while masked by the arbitrary contents of the catalytic tape τ. Similarly, cryptography shows how to privately outsource a computation to untrusted parties by masking the computation's inputs with randomness. We are optimistic that decades of research in multi-party computation may broaden what is possible in the realm of catalytic algorithms.

2 Technical Overview

2.1 Tree Evaluation and the Cook–Mertz Algorithm

We begin with an overview of the Cook–Mertz algorithm for tree evaluation, following the presentation of Goldreich [Gol25]. We first formally define the tree evaluation problem:

Definition 2.1 (TreeEval). *For positive integers h, ℓ, the problem $\mathsf{TreeEval}_{h,\ell}$ is the following task [CMW+12]: given a complete binary tree of height h, where*

- *each leaf node is indexed by $u \in \{0,1\}^h$ and labeled by a value $v_u \in \{0,1\}^\ell$ and*
- *each internal node is indexed by $u \in \{0,1\}^{<h}$ and labeled by a function $f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$,*

evaluate the tree in a bottom-up manner by setting each internal node's value to be the evaluation of its function on its children's values, and output the value of the root node. In other words, output the value $v_\emptyset$ such that $v_u = f_u(v_{u0}, v_{u1})$ for all $u \in \{0,1\}^{<h}$. The input length to the $\mathsf{TreeEval}_{h,\ell}$ problem is thus $n = \Theta(2^h \cdot \ell \cdot 2^{2\ell})$, so we have $h, \ell \le O(\log n)$.

Definition 2.2 (Higher-arity TreeEval). *For an integer $r \ge 2$, we define $\mathsf{TreeEval}_{h,\ell,r}$ to be the analogous problem with a complete r-ary tree. Thus, $\mathsf{TreeEval}_{h,\ell}$ is equivalent to $\mathsf{TreeEval}_{h,\ell,2}$.*

For clarity, we focus in this work on the arity $r = 2$ case; Lemma 3.6 gives a simple reduction from TreeEval for any constant r to TreeEval for $r = 2$ that suffices for the application of [Wil25].

A baseline algorithm. A simple algorithm for TreeEval proceeds using a depth-first recursion to evaluate the value of each node, from leaves all the way up to the root. On input length $n \in \mathbb{N}$, tree height $h \in \mathbb{N}$, and value bitlength $\ell \in \mathbb{N}$, this solution runs in $\mathsf{poly}(n)$ time and uses space $O(h\ell) = O(\log^2 n)$. The bottleneck leading to this space cost is that, as the depth-first recursion proceeds all the way down the height-h tree, the algorithm must hold up to $O(h)$ internal nodes' values along a path from the root, each of which comprises ℓ bits. Indeed, TreeEval was posed in part as a candidate for separating the class L of all logarithmic space computations from the class P of all polynomial-time computations on exactly this intuition; it was considered plausible that roughly $h\ell = O(\log^2 n)$ bits of space are necessary, until the algorithms of Cook and Mertz [CM20, CM24] shattered this barrier.

The Cook–Mertz algorithm as a one-level gadget. The algorithm of Cook and Mertz [CM24] builds on this baseline $O(h\ell)$-space recursive algorithm yet shrinks its memory footprint by reusing the same $o(h\ell)$ bits of space to hold a path of values from the root node. This seemingly-impossible task is accomplished using techniques from catalytic computation: instead of computing a new internal node value by writing it into fresh space, the algorithm computes the value by adding a multiple of it into an existing register. To be more precise, the improved algorithmic task at the core of Cook-Mertz [CM24] can be distilled as the following subroutine, which we informally refer to as the "one-level gadget":

Definition 2.3 (Informal definition of the one-level gadget). *Let m be a positive integer and $\mathcal{R}$ a commutative ring containing $\mathbb{Z}_m$. For an element $x \in \mathcal{R}$, we define the oracle $\mathcal{O}_x(\tilde{\tau} \in \mathcal{R}, \alpha \in \mathbb{Z}_m)$ to be that which updates a register $\tilde{\tau} \leftarrow \tilde{\tau} + \alpha x$ in place.*

For a positive integer ℓ and a canonical encoding $\mathsf{Enc} : \{0,1\}^\ell \to \mathcal{R}$—which, like every object in this paper, the algorithm must be able to generate entry by entry in small space (as made precise in Remark 3.8)—the problem $\mathsf{OneLevelGadget}_{\ell,\mathcal{R},\mathsf{Enc}}$ is the following task: given as input

- *three registers $\tau_0, \tau_1, \tau \in \mathcal{R}$,*
- *oracle access to $\mathcal{O}_{\mathsf{Enc}(x_0)}$ and $\mathcal{O}_{\mathsf{Enc}(x_1)}$ for some unknown strings $x_0, x_1 \in \{0,1\}^\ell$,*
- *the truth table of a function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$, and*
- *a scalar $\alpha^* \in \mathbb{Z}_m$,*

perform the update $\tau \leftarrow \tau + \alpha^* \cdot \mathsf{Enc}(f(x_0, x_1))$ while leaving the registers τ_0 and τ_1 unchanged at the end of the computation (though they may be temporarily modified throughout).

Intuitively, the one-level gadget builds the oracle $\mathcal{O}_{\mathsf{Enc}(f(x_0,x_1))}$ from both oracles $\mathcal{O}_{\mathsf{Enc}(x_0)}$ and $\mathcal{O}_{\mathsf{Enc}(x_1)}$. Cook–Mertz then show that the gadget can be composed with itself recursively at low overhead in space.

Specifically: if the OneLevelGadget algorithm makes Q oracle queries, uses $O(\ell)$ bits of workspace between queries, of which only ℓ' bits are retained across queries, and runs in time $\mathsf{poly}(2^\ell)$, then the recursion keeps one ℓ'-bit slot for each call on its call stack (at most one per level of the tree) plus a single shared $O(\ell)$-bit block of scratch space. The resulting $\mathsf{TreeEval}_{h,\ell}$ algorithm uses

1. $O(\ell + h \cdot \ell')$ bits of *free* space, as well as
2. the three aforementioned registers, which consist of $O(\log |\mathcal{R}|)$ bits, persist through the whole computation, and are what the algorithm treats as its *catalytic* space.

The runtime is $\mathsf{poly}(Q^h, 2^\ell)$. (One should expect $\ell' \geq \log Q$, since the gadget must at least know which query it is up to.) Theorems 4.1 and 4.4 give precise versions of these statements.

Looking ahead, our algorithm also follows this template: in Section 4.1 we construct a new algorithm for the one-level gadget and in Section 4.3 we lift this to an algorithm for TreeEval. Our innovation is in the one-level-gadget component, which we focus on for the remainder of the technical overview.

Constructing Cook–Mertz's one-level gadget. Next, we outline Cook-Mertz's one-level gadget, using the presentation of Goldreich (and omitting some bookkeeping details to follow in the body of the paper) [Gol25].

First, some setup: let $\mathbb{F}$ be a finite field of prime order $p \in (2\ell + 2, 4\ell + 4)$. (One exists, by Bertrand's postulate.) Set $k := |\mathbb{F}| - 1 > 2\ell$, and let ω be a generator of $\mathbb{F}^*$, i.e., a primitive k^{th} root of unity. Define the ring $\mathcal{R} = \mathbb{F}^\ell$ and the encoding function $\mathsf{Enc} : \{0,1\}^\ell \to \mathcal{R}$ as $\mathsf{Enc}(x) = x$, viewing each entry of the string $x \in \{0,1\}^\ell$ as an element of the field $\mathbb{F}$ in the natural way.

Given as inputs the three registers τ_0, τ_1, and $\tau \in \mathcal{R}$, the function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$, and the scalar $\alpha^* \in \mathbb{Z}_m$, the Cook-Mertz gadget then proceeds as follows:

1. Let $\hat{f} : \mathbb{F}^\ell \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ be the multilinear extension of the function f. Define the function $g : \mathbb{F} \to \mathbb{F}^\ell$ by $g(x) := \hat{f}(x\tau_0 + x_0, x\tau_1 + x_1)$. Note that g has degree at most $2\ell < k$ and that the gadget's goal is to add the value $\alpha^* \cdot g(0)$ into the register τ. By Lagrange interpolation, it holds that:

$$g(0) = \frac{1}{k} \sum_{i=0}^{k-1} g(\omega^i). \tag{1}$$

2. Repeat for each $i = 0, 1, \ldots, k-1$:

   (a) Using queries to the oracles $\mathcal{O}_{\mathsf{Enc}(x_0)}$ and $\mathcal{O}_{\mathsf{Enc}(x_1)}$, update $\tau_0 \leftarrow \tau_0 + \omega^{-i} x_0$ and $\tau_1 \leftarrow \tau_1 + \omega^{-i} x_1$.

   (b) Update $\tau \leftarrow \tau + \alpha^*/k \cdot \hat{f}(\omega^i \tau_0, \omega^i \tau_1)$.[2]

   (c) Using queries to the oracles $\mathcal{O}_{\mathsf{Enc}(x_0)}$ and $\mathcal{O}_{\mathsf{Enc}(x_1)}$ again, restore $\tau_0 \leftarrow \tau_0 - \omega^{-i} x_0$ and $\tau_1 \leftarrow \tau_1 - \omega^{-i} x_1$.

Correctness of this algorithm follows from equation (1): the algorithm indeed performs the desired update $\tau \leftarrow \tau + \alpha^* \cdot \mathsf{Enc}(f(x_0, x_1))$, while leaving the registers τ_0 and τ_1 unchanged at the end of the computation.

In terms of efficiency, this gadget makes $4k = O(\ell)$ oracle queries; the only state it retains across queries is the counter i, the scalar α^*, and its position in the loop: in sum, $\ell' = O(\log \ell)$ bits. As a result, the corresponding $\mathsf{TreeEval}_{h,\ell}$ algorithm uses free space $O(\ell + h\ell') = O(\ell + h \log \ell)$, catalytic space $O(\log |\mathcal{R}|) = O(\ell \log \ell)$, and runtime $\mathsf{poly}(k^h, 2^\ell) = \mathsf{poly}(\ell^h, 2^\ell)$. In this parameter regime, the "catalytic" registers occupy only $O(\ell \log \ell)$ bits so nothing is lost by folding them into the free space, as stated in Corollaries 4.3 and 4.9. (We describe these registers as catalytic here to ease the reader into the presentation; in our main results, the catalytic space will become more important and super-polylogarithmic in size.)

[2] The algorithm never writes down $\hat{f}$: the update is computed in one streaming pass over the truth table of f, evaluating the multilinear extension term-by-term and coordinate-by-coordinate and accumulating.

2.2 This Work: A Connection between TreeEval and Private Information Retrieval

Our improved algorithm for OneLevelGadget takes inspiration from cryptography. Its starting point is the observation that the univariate polynomial-interpolation trick in Cook–Mertz's one-level gadget appears almost verbatim in private information retrieval, as we describe next. This view is key to our results, which leverage an arsenal of techniques from private-information-retrieval to speed up OneLevelGadget and, as a result, TreeEval.

A primer on private information retrieval. A private-information-retrieval (PIR) protocol [CGKS95, CGKS98, KO97] allows a user to read a bit from a remote database $\mathsf{DB} \in \{0,1\}^{n_{\mathsf{DB}}}$, of which a copy is stored on $k \geq 2$ servers, without revealing to any of the servers what bit was read. In more detail, a PIR protocol consists of three algorithms:

1. $\mathsf{Query}(i \in [n_{\mathsf{DB}}]) \to \mathsf{qu}_0, \ldots, \mathsf{qu}_{k-1}$, which the user runs on the index $i \in [n_{\mathsf{DB}}]$ that it wants to read and which outputs k PIR queries, each of which is sent to one of the k PIR servers.

2. $\mathsf{Answer}(\mathsf{DB} \in \{0,1\}^{n_{\mathsf{DB}}}, \mathsf{qu}) \to \mathsf{ans}$, which each server runs on the database $\mathsf{DB} \in \{0,1\}^{n_{\mathsf{DB}}}$ and on the PIR query it received and which outputs a PIR answer.

3. $\mathsf{Reconstruct}(i \in [n_{\mathsf{DB}}], \mathsf{ans}_0, \ldots, \mathsf{ans}_{k-1}) \to \{0,1\}$, which the user runs on its index $i \in [n_{\mathsf{DB}}]$ and on the list of k PIR answers from all servers and which then outputs the recovered database bit.

Taken together, these three algorithms satisfy two requirements: *correctness* and *privacy*. The correctness requirement guarantees that, if it was executed honestly, the user indeed recovers the i^{th} bit of the database DB at the end of the protocol. The privacy requirement guarantees that the marginal distribution of each query qu_j, for $j \in [k]$, is independent of the index i being queried. In almost all information-theoretic PIR schemes [CGKS95, BIM00, WY05, GLM+25, HR26], this is achieved by additively masking the index i, encoded as an element of some vector space, with uniformly-sampled randomness.

In terms of PIR efficiency, two particular metrics are relevant to this work: (1) the number of required servers $k \geq 2$, and (2) the scheme's *communication complexity* $\mathsf{CC}(n_{\mathsf{DB}})$, defined as the maximal bitlength of the k PIR queries $\mathsf{qu}_1, \ldots, \mathsf{qu}_k$ and the k PIR answers $\mathsf{ans}_1, \ldots, \mathsf{ans}_k$ on a worst-case, n_{DB}-bit database.

Background: PIR from polynomial interpolation. A classic "Reed-Muller PIR" scheme builds on multilinear extensions and univariate polynomial interpolation [CGKS95, BIM00, WY05], much like the Cook–Mertz gadget we just saw. In this PIR scheme, the user encodes its index $i \in [n_{\mathsf{DB}}]$ into a point in a vector space, then shoots a random line through this point, and sends each PIR server a point on this line. The k PIR servers evaluate a multilinear extension $\hat{f}$, which encodes the database DB, at the point they received and send the result back to the user. By design, these PIR answers trace out a low-degree univariate polynomial; reconstruction interpolates that polynomial to recover the desired value DB_i.

Put in the same language we just used to describe Cook–Mertz, the steps in this PIR scheme are:

- The servers interpret the n_{DB}-bit database DB as a multilinear polynomial $\hat{f} : \{0,1\}^{\ell} \to \{0,1\}$, where $\ell = O(\log n_{\mathsf{DB}})$ and the polynomial's evaluations at a chosen set of points match the database values.

  Let $\mathbb{F}$ be a finite field of prime order in $(\ell+2, 2\ell+4)$, let $k := |\mathbb{F}| - 1 > \ell$, and let ω be a generator of $\mathbb{F}^*$, i.e., a primitive k^{th} root of unity.

- $\mathsf{Query}(i \in [n_{\mathsf{DB}}])$: Parse the index i as a vector $x = x(i) \in \{0,1\}^{\ell} \in \mathbb{F}^{\ell}$. Sample a random mask $\tau \xleftarrow{R} \mathbb{F}^{\ell}$. For server $j \in [0, k-1]$, output $\mathsf{qu}_j = \tau + \omega^{-j} \cdot x$.

Define the univariate polynomial $g : \mathbb{F} \to \mathbb{F}$ to be the restriction of the database polynomial to the line spanned by the query points: i.e., $g(y) = \hat{f}(y\tau + x)$. The polynomial g has degree $\leq \ell$ and the user's goal is to learn $g(0)$, which is equal to the i^{th} database bit.

- $\mathsf{Answer}(\mathsf{DB}, \mathsf{qu})$: Server j replies with $\mathsf{ans}_j = \hat{f}(\omega^j \mathsf{qu})$.
- $\mathsf{Reconstruct}(i, \mathsf{ans}_0, \ldots, \mathsf{ans}_{k-1})$: The user computes $\frac{1}{k}\sum_{j=0}^{k-1} \mathsf{ans}_j$, which is $g(0) = \mathsf{DB}_i$ by equation (1).

This view reveals a surprising connection: the Cook-Mertz OneLevelGadget for TreeEval [CM24] essentially evaluates the function f by making white-box use of a k-server PIR protocol.

Connecting TreeEval to PIR. Recall the task in the OneLevelGadget: the algorithm must evaluate the function $f(x_0, x_1)$—or, equivalently, retrieve the record at index $x_0 \| x_1$ from the truth table of f—without seeing either of the input strings x_0 or x_1 in the clear. Its only access to these inputs is through the oracles $\mathcal{O}_{\mathsf{Enc}(x_0)}$ and $\mathcal{O}_{\mathsf{Enc}(x_1)}$, which add multiples of their encodings into the registers τ_0 and τ_1. After doing so, the values x_0 and x_1 appear masked by the registers' existing contents.

This is the same challenge faced by a PIR protocol, which aims to read the bit at index i from a database while exposing only additively masked copies of i. With this in mind, we set up an informal dictionary between the two settings in Table 2. For Reed–Muller PIR, $c_j = \omega^{-j}$, Enc is the identity, and server j answers $\hat{f}(\omega^j \mathsf{qu}_j)$; the gadget column then becomes exactly the Cook–Mertz update $\tau \leftarrow \tau + (\alpha^*/k)\, \hat{f}(\omega^j \tau_0, \omega^j \tau_1)$ above. This mapping from private information retrieval to catalytic computation of a function f is intended as an intuitive (rather than a formal) correspondence. Making it precise requires modifying the algorithm and imposing a number of structural requirements on the PIR scheme, which we describe informally next:

1. *Composability.* PIR is a one-shot primitive: the user makes a PIR query, receives PIR answers, and recovers a database record, each of which may have its own type. The one-level gadget, on the other hand, is recursively composed with itself as part of the TreeEval algorithm. As a result, its output oracle needs to take exactly the form that its input oracles provide—an encoding added into a register—because the parent register τ plays the role of a child register one level up in the recursive TreeEval algorithm. Answers, in other words, must have the same type as queries. Reed–Muller PIR happens to satisfy this: the encoding is the identity, and queries and answers are both vectors in $\mathbb{F}^\ell$. For other PIR schemes it fails outright, and resolving it while also addressing the below bullet is one of our main challenges.

2. *Reconstruction is blind.* In PIR, the Reconstruct algorithm receives the index i and all k servers' answers at once. The OneLevelGadget is far more constrained: its input is stored in catalytic space (i.e., masked by the arbitrary register contents), and there is no free space available to store even a single answer. Reconstruction must therefore "stream" through the servers' answers one at a time, adding them into the same registers and outputting the reconstructed database entry at the end (again in catalytic space, i.e., masked by arbitrary values).

   One natural way of ensuring this is to require that the PIR scheme's reconstruction procedure applies a predetermined linear combination to each answer as it is produced.

3. *The index arrives in two halves.* In PIR, the Query algorithm receives as input the index i. In the OneLevelGadget, the inputs x_0 and x_1 are instead encoded by separate oracles into separate registers, so a PIR query for the pair (x_0, x_1) must be assembled from independent PIR queries for each half (fetched by previous recursive calls, as per point (1)).

k-server PIR	One-level gadget
Entities in protocol	
Database DB	Truth table of the function f
Index i to read from DB	The input (x_0, x_1) at which to evaluate f
PIR client and k PIR servers	One algorithm, playing every role in sequence
Computations	
PIR query to server j: $\mathsf{qu}_j = \tau + c_j \cdot \mathsf{Enc}(i)$, for • a mask τ sampled uniformly at random, • a server-dependent scalar c_j, and • an encoding $\mathsf{Enc}(i)$ of the index i.	Contents of register τ_σ ($\sigma \in \{0,1\}$) in round j, set by oracle calls: $\tau_\sigma + c_j \cdot \mathsf{Enc}(x_\sigma)$, for • the arbitrary initial register contents τ_σ, • a scalar c_j chosen by the algorithm, and • the encoding $\mathsf{Enc}(x_\sigma)$ of the input string x_σ.
PIR answer ans_j, computed by server j from DB and qu_j	One streaming pass over the truth table of f, computing the same function of the current contents of τ_0, τ_1 and adding it into register τ
PIR reconstruction: user takes a fixed linear combination of the PIR answers	The same linear combination, accumulated in τ as the passes complete
Cost metrics	
Number of servers	Number of oracle calls
Communication complexity $\mathsf{CC}(n_{\mathsf{DB}})$	Catalytic space: the size of the registers τ_0, τ_1, τ

Table 2: An informal dictionary between k-server PIR, for schemes whose queries have the form "mask plus a server-dependent multiple of an encoding of the index", and the one-level gadget.

One way of overcoming this issue is by requiring the PIR scheme's queries to be "concatenation-friendly": that is a PIR query for the pair $x_0 \| x_1$ is the concatenation of the PIR query for x_0 and x_1. As we will see in Section 4, while it is natural for the Reed-Muller PIR described above, massaging other PIR schemes to satisfy this property is more troublesome.

4. *The catalytic tape is arbitrary.* PIR usually masks queries uniformly at random, while in $\mathsf{OneLevelGadget}$ we must work with a possibly adversarially chosen catalytic tape as the mask. These can be unified by simply restricting attention to PIR schemes that have perfect correctness. We will also make the structural requirement that the PIR queries be of the form "mask plus a fixed encoding of the index."

2.3 Adapting Matching-Vector PIR to $\mathsf{TreeEval}$

Despite the aforementioned technical differences, the dictionary in Table 2 is a useful tool for anticipating what $\mathsf{TreeEval}$ efficiency a PIR protocol might offer. In the language of the introduction (Table 1), the number of servers becomes the decoder's sparsity, and the communication becomes the matching-vector dimension (times $\log m$). In particular, the algorithm of Cook and Mertz [CM24] owes its low catalytic space to the

low communication of Reed–Muller PIR, which achieves communication complexity $\tilde{O}(\log n_{\mathsf{DB}})$ when instantiated with $\tilde{O}(\log n_{\mathsf{DB}})$ servers. However, that same $O(\log n_{\mathsf{DB}})$-server parameter regime results in the Cook-Mertz algorithm requiring (1) superpolynomial runtime and (2) slightly superlogarithmic free space, since this super-constant number of servers corresponds to a super-constant number of recursive oracle calls at each internal node. The dictionary also explains the earlier one-hot algorithm of Cook and Mertz [CM20, Theorem 1]: it corresponds to the four-server XOR ("cube") scheme of [CGKS95], whose constant number of servers yields $O(1)$ recursive calls per level, and hence $O(h + \ell)$ free space and polynomial time, but whose $\sqrt{n_{\mathsf{DB}}}$ communication yields polynomial catalytic space $O(2^{\ell})$.

Which PIR scheme to use? With this in mind, the PIR literature points us to a different PIR protocol that offers hope for a new TreeEval algorithm, achieving a new tradeoff between runtime, free space, and catalytic space: *matching-vector PIR* [Efr09, DG16, GKS25]. This family of PIR schemes, which gives rise to the most communication-efficient PIR known to date, needs only a *constant* number of servers to achieve already-*subpolynomial* communication: namely, communication $\exp(\tilde{O}((\log n_{\mathsf{DB}})^{1/t}))$ bits using 2^t servers, for any constant $t \in \mathbb{N}$.

Using matching-vector PIR, the dictionary in Table 2 thus predicts a TreeEval algorithm with free space $O(h + \ell)$, polynomial runtime, and subpolynomial catalytic space $\exp(\tilde{O}(\ell^{1/t}))$. Making this scheme work is the goal of the remainder of this paper and what ultimately proves Theorem 1.1.

Efremenko's matching-vector PIR. We begin with a review of matching-vector PIR. Given a modulus m (to be specific later on), let $\mathbb{F}$ be a finite field containing a primitive m^{th} root of unity γ. Then, the scheme of [Efr09] is built from two related objects:

Definition 2.4 (S-matching vectors)**.** *Let $S \subseteq \mathbb{Z}_m$ be a set containing 0. The vectors $\mathbf{u}_1, \dots, \mathbf{u}_N$ and $\mathbf{v}_1, \dots, \mathbf{v}_N \in \mathbb{Z}_m^d$ form an* S-matching-vector family *if their inner products are constrained in the following way:*

- $\langle \mathbf{u}_i, \mathbf{v}_i \rangle = 0$ *for every* $i \in [N]$*, and*
- $\langle \mathbf{u}_i, \mathbf{v}_j \rangle \in S \backslash \{0\}$ *for every* $i \neq j \in [N]$.

Definition 2.5 (A-decoding polynomial)**.** *Let $A \subseteq \mathbb{Z}_m$ be a set containing 0. A polynomial $P \in \mathbb{F}[X]$ is an* A-decoding polynomial *if its evaluations are constrained in the following way:*

- $P(1) = 1$*, and*
- $P(\gamma^j) = 0$ *for every* $j \in A \backslash \{0\}$.

The sparsity *of P is its number of monomials, i.e., the integer k when we write $P(X) = \sum_{j=0}^{k-1} c_j X^{b_j}$ with distinct exponents and nonzero coefficients.*

For now, take $A = S$. Suppose a user wants to learn the ith entry of a database $\mathsf{DB} \in \mathbb{F}^{n_{\mathsf{DB}}}$; then we can obtain Efremenko's scheme by working backwards from the goal:

$$
\begin{aligned}
\mathsf{DB}_i &= \sum_{y \in [n_{\mathsf{DB}}]} \mathsf{DB}_y \cdot \mathbb{1}[i = y] \\
&= \sum_{y \in [n_{\mathsf{DB}}]} \mathsf{DB}_y \cdot \mathbb{1}\left[\langle \mathbf{u}_i, \mathbf{v}_y \rangle = 0\right] && \text{(matching vector property)} \\
&= \sum_{y \in [n_{\mathsf{DB}}]} \mathsf{DB}_y \cdot P\left(\gamma^{\langle \mathbf{u}_i, \mathbf{v}_y \rangle}\right) && (P \text{ is } S\text{-decoding}) \\
&= \sum_{y \in [n_{\mathsf{DB}}]} \mathsf{DB}_y \sum_{j=0}^{k-1} c_j \, \gamma^{b_j \langle \mathbf{u}_i, \mathbf{v}_y \rangle} && \text{(expand } P\text{)}
\end{aligned}
$$

$$= \sum_{j=0}^{k-1} c_j \left(\sum_{y \in [n_{\mathsf{DB}}]} \mathsf{DB}_y \cdot \gamma^{\langle b_j \mathbf{u}_i, \mathbf{v}_y \rangle} \right). \qquad \text{(swap the sums)}$$

The PIR scheme works by indexing the servers by $j \in [k]$ and letting server j be responsible for the corresponding inner sum over all $y \in [n_{\mathsf{DB}}]$ in the above. While the user could recover DB_i by sending server j the single vector $\mathsf{qu}_j = b_j \mathbf{u}_i$, this would reveal the user's own index i and thus not be private.

To ensure privacy, Efremenko simply adds a mask: the user samples $\mathbf{z} \xleftarrow{\mathrm{R}} \mathbb{Z}_m^d$ uniformly at random and instead sends to server j the query $\mathsf{qu}_j = \mathbf{z} + b_j \mathbf{u}_i$, which is marginally uniform. Each summand above then picks up the factor $\gamma^{\langle \mathbf{z}, \mathbf{v}_y \rangle}$, and the same calculation shows that the answers $\mathsf{ans}_j = \sum_y \mathsf{DB}_y \cdot \gamma^{\langle \mathsf{qu}_j, \mathbf{v}_y \rangle}$ reconstruct the *masked* record:

$$\gamma^{\langle \mathbf{z}, \mathbf{v}_i \rangle} \cdot \mathsf{DB}_i = \sum_{j=1}^{k} c_j \cdot \mathsf{ans}_j. \tag{2}$$

As he knows $\mathbf{z}$, the user computes $\langle \mathbf{z}, \mathbf{v}_i \rangle$ and divides by the mask, thus recovering the desired entry DB_i. In short, given an S-matching vector family of size n_{DB} over $\mathbb{Z}_m^d$ and an S-decoding polynomial of sparsity k, the k-server PIR protocol by [Efr09] achieves communication $O(d \log m)$ per server.

Finally, to obtain the required mathematical objects, state-of-the-art constructions take m to be a product of t distinct, odd primes and set $S := S_m^*$ to be the *canonical set*, which comprises the 2^t residues that are congruent to 0 or 1 modulo each prime factor of m. Then, Lagrange interpolation gives a decoding polynomial of degree $\leq |S| - 1$ and therefore sparsity $k \leq |S| = 2^t$, and Grolmusz's construction [Gro00, BBR94] gives an S_m^*-matching vector family of size N with dimension $d = \exp(\tilde{O}((\log N)^{1/t}))$. When t and m are constant, this yields PIR with a constant number of servers and subpolynomial communication.

A one-level gadget from matching-vector PIR. We now translate this scheme to a OneLevelGadget for tree evaluation, proceeding with the informal dictionary of Table 2. First, we index the matching-vector family by $\{0,1\}^\ell$, so that each value $x \in \{0,1\}^\ell$ has vectors $\mathbf{u}_x, \mathbf{v}_x \in \mathbb{Z}_m^d$, and we take $\mathcal{R} = \mathbb{Z}_m^d$ with $\mathsf{Enc}(x) = \mathbf{u}_x$; the oracle $\mathcal{O}_{\mathsf{Enc}(x_b)}$ then adds multiples of $\mathbf{u}_{x_b}$ into the register $\tau_b \in \mathbb{Z}_m^d$. For now, we regard the outputs of the function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$ as elements of $\mathbb{F}$, with the parent register $\tau \in \mathbb{F}$.

Additionally, we assume that $S \cap (-S) = \{0\}$, so that a sum of two permitted inner products vanishes only if both do; this holds for S_m^* whenever m is odd. Correspondingly, we take the polynomial P to be A-decoding for the sumset $A = S + S := \{s + s' : s, s' \in S\}$. This then allows for the following computation:

$$\begin{aligned}
f(x_0, x_1) &= \sum_{y_0, y_1 \in \{0,1\}^\ell} f(y_0, y_1) \cdot \mathbb{1}[(y_0, y_1) = (x_0, x_1)] \\
&= \sum_{y_0, y_1} f(y_0, y_1) \cdot \mathbb{1}\left[\langle \mathbf{u}_{x_0}, \mathbf{v}_{y_0} \rangle + \langle \mathbf{u}_{x_1}, \mathbf{v}_{y_1} \rangle = 0 \right] && \text{(matching vectors, } S \cap (-S) = \{0\}) \\
&= \sum_{y_0, y_1} f(y_0, y_1) \cdot P\left(\gamma^{\langle \mathbf{u}_{x_0}, \mathbf{v}_{y_0} \rangle + \langle \mathbf{u}_{x_1}, \mathbf{v}_{y_1} \rangle} \right) && (P \text{ is } (S{+}S)\text{-decoding}) \\
&= \sum_{y_0, y_1} f(y_0, y_1) \sum_{j=1}^{k} c_j \, \gamma^{b_j \langle \mathbf{u}_{x_0}, \mathbf{v}_{y_0} \rangle + b_j \langle \mathbf{u}_{x_1}, \mathbf{v}_{y_1} \rangle} && \text{(expand } P) \\
&= \sum_{j=1}^{k} c_j \left(\sum_{y_0, y_1} f(y_0, y_1) \, \gamma^{\langle b_j \mathbf{u}_{x_0}, \mathbf{v}_{y_0} \rangle + \langle b_j \mathbf{u}_{x_1}, \mathbf{v}_{y_1} \rangle} \right). && \text{(swap the sums)}
\end{aligned}$$

Each inner sum can be computed in one streaming pass over the truth table of the function f, given the pair of queries $(b_j \mathbf{u}_{x_0}, b_j \mathbf{u}_{x_1})$.

Finally, masking works much in the same way as in Efremenko's PIR: we do not have space to write down the queries $b_j \mathbf{u}_{x_0}$ and $b_j \mathbf{u}_{x_1}$, so instead we store them in catalytic registers, masking them with the pre-existing contents (τ_0, τ_1) of the catalytic tape. Now, two oracle calls update $\tau_0 \leftarrow \tau_0 + b_j \mathbf{u}_{x_0}$ and $\tau_1 \leftarrow \tau_1 + b_j \mathbf{u}_{x_1}$. Each summand picks up the factor $\gamma^{\langle \tau_0, \mathbf{v}_{y_0} \rangle + \langle \tau_1, \mathbf{v}_{y_1} \rangle}$, and the analogue of equation (2) reads

$$\begin{aligned} \gamma^{\langle \tau_0, \mathbf{v}_{x_0} \rangle + \langle \tau_1, \mathbf{v}_{x_1} \rangle} \cdot f(x_0, x_1) &= \sum_{j=1}^{k} c_j \cdot \mathsf{ans}_j, \\ \text{where} \quad \mathsf{ans}_j &= \sum_{y_0, y_1} f(y_0, y_1)\, \gamma^{\langle \tau_0 + b_j \mathbf{u}_{x_0}, \mathbf{v}_{y_0} \rangle + \langle \tau_1 + b_j \mathbf{u}_{x_1}, \mathbf{v}_{y_1} \rangle}, \end{aligned} \tag{3}$$

with the same coefficients c_j as in equation (2). Here, the coefficients c_j are fixed in advance, so the reconstruction is indeed blind as we wanted in Section 2.2. To handle the correction factor of $\gamma^{\langle \tau_0, \mathbf{v}_{x_0} \rangle + \langle \tau_1, \mathbf{v}_{x_1} \rangle}$, the gadget separately computes this quantity using additional oracle calls and a standard catalytic algorithm for computing inner products (for which the oracles must also provide the partner vectors $\mathbf{v}_{x_b}$; see Section 4.1). In total, the gadget makes $O(k)$ oracle calls and keeps $O(\log k)$ bits for bookkeeping, with catalytic registers of $O(d \log m)$ bits. For Grolmusz's families over a constant number of primes, k and m are constants and $d = \exp(\tilde{O}(\ell^{1/t}))$, so once the gadget is made recursively composable (below), these costs yield exactly the algorithm the dictionary predicted: free space $O(h + \ell)$, polynomial time, and catalytic space $\exp(\tilde{O}(\ell^{1/t}))$, the last row of Table 1.

Remark 2.6. *A natural alternative would avoid the assumption $S \cap (-S) = \{0\}$ by decoding each child separately: apply an S-decoding polynomial to each of the two indicators and multiply, at the cost of k^2 streaming passes indexed by pairs of monomials. This alternative also works for our main result; the benefit is that it only needs an S-decoding polynomial rather than an $(S+S)$-decoding polynomial, while the cost is that it requires more recursive calls—$O(k^2)$ to loop independently for each child, rather than the $O(k)$ we achieve with the sumset setup by looping "in tandem". These distinctions do not matter for our main result; however, the sumset route is better suited for recovering the $O(\log n \log\log n / \log\log\log n)$ space algorithm of Stoeckl [Sto23] and Goldreich [Gol24], as we explain at the end of this section; Remark 4.13 makes the comparison precise.*

Ensuring recursive composability. One issue remains: to compose the gadget recursively, the registers τ_0, τ_1, and τ must all have the same type, since the parent's register serves as a child's register one level up. This is currently not the case, as $\tau_0, \tau_1 \in \mathbb{Z}_m^d$ while $\tau \in \mathbb{F}$.

At first glance, this may seem like an innocent technicality, but this is actually a serious obstacle. To fix it, we would need a commutative ring $\mathcal{R}$ that simultaneously contains $\mathbb{Z}_m$ (i.e., $\mathcal{R}$ has characteristic exactly m) and also has an mth root of unity γ. This prevents $\mathcal{R}$ from being a finite field; however, one could hope to take $\mathcal{R}$ to be a ring such as $\mathbb{Z}_m[X]/(X^m - 1)$, the ring in which matching-vector codes are usually presented [DGY11, Efr09]. However, it turns out that any such ring *cannot* have an A-decoding polynomial for our set $A = S + S$, as the below lemma states. (Here, we are reinterpreting the definition of A-decoding polynomials to have outputs in $\mathcal{R}$ in the natural way.)

Lemma 2.7. *Let $\mathcal{R}$ be a commutative ring (with a multiplicative identity element) of characteristic exactly $m \geq 2$, and let $A \subseteq \mathbb{Z}_m$ be a set containing an element whose additive order in $\mathbb{Z}_m$ is a prime power p^a with $a \geq 1$. Then, there do not exist $\gamma \in \mathcal{R}$ and $P \in \mathcal{R}[X]$ satisfying*

$$\gamma^m = 1, \qquad P(1) = 1, \qquad P(\gamma^j) = 0 \quad \text{for every } j \in A \backslash \{0\}.$$

We defer the proof to Section A. For $S = S_m^*$, the hypothesis of the lemma is met: the residue that is 1 modulo p_1 and 0 modulo every other prime factor of m lies in $S_m^* \subseteq S + S$ and has additive order p_1, a

prime. To bypass this issue, we work with an alternate definition that gives us the same functionality as an A-decoding polynomial but additionally has type-consistency baked into it:

Definition 2.8 (A-decoder)**.** *Let $A \subseteq \mathbb{Z}_m$ be a set containing 0. An A-decoder over $\mathbb{Z}_m$ of sparsity k is a tuple of functions $\varphi_1, \dots, \varphi_k : \mathbb{Z}_m \to \mathbb{Z}_m$ and elements $b_1, \dots, b_k \in \mathbb{Z}_m$ such that:*

- $\sum_{j=1}^{k} \varphi_j(z + ab_j) = 0$ *for every $a \in A \backslash \{0\}$ and every $z \in \mathbb{Z}_m$, and*
- $\sum_{j=1}^{k} \varphi_j(0) = 1$.

The two conditions above mirror the two defining properties of an A-decoding polynomial, and setting $\varphi_j(y) = c_j \gamma^y$ recovers essentially the guarantees of one—except that these φ_j take values in $\mathbb{F}$ rather than $\mathbb{Z}_m$. (Lemma 3.18 records the exact correspondence with a weakened, ring-valued variant of the definition.) We construct $(S{+}S)$-decoders satisfying the full definition over $\mathbb{Z}_m$, with sparsity 3^t for $S = S_m^*$, in Lemma 3.19. With an $(S{+}S)$-decoder in place of P, the gadget of the previous paragraph goes through essentially unchanged, now with all three registers of type $\mathbb{Z}_m^d$. Instantiated with Grolmusz's matching vectors, this yields a TreeEval algorithm with free space $O(h + \ell) = O(\log n)$, catalytic space $2^{O(\log^{\epsilon} n)}$, and $\mathsf{poly}(n)$ runtime (Corollary 4.7), proving Theorem 1.1.

Alternative instantiations matching Cook–Mertz [CM24] and improvements [Sto23, Gol24]. Our framework has a second interesting regime: instead of Grolmusz's matching vectors over a product of t primes, we can use a single odd prime p together with a denser family of matching vectors whose inner products fall in a larger set S. This recovers the guarantees of Cook–Mertz [CM24] and the $\log \log \log n$ factor improvement of Stoeckl [Sto23] and Goldreich [Gol24]. We outline this construction below.

To define the matching vectors, recast strings $x \in \{0,1\}^{\ell}$ as elements of $[M]^w$, for a coordinate bound M and width w to be chosen shortly (the only constraint is that $M^w \geq 2^{\ell}$). Then define

$$\mathbf{u}_x = \left(1, x_1, \dots, x_w, \sum_{j=1}^{w} x_j^2\right), \qquad \mathbf{v}_y = \left(\sum_{j=1}^{w} y_j^2, -2y_1, \dots, -2y_w, 1\right), \tag{4}$$

vectors of dimension $d = w + 2$, so that

$$\langle \mathbf{u}_x, \mathbf{v}_y \rangle = \sum_{j=1}^{w} (x_j - y_j)^2.$$

As an integer, this score is nonnegative, at most $D := wM^2$, and zero exactly when $x = y$. Now choose the modulus m to be a prime in $(2D, 4D)$ and view the vectors in $\mathbb{Z}_m$. The inner products lie in the set $S = \{0, 1, \dots, D\} \subseteq \mathbb{Z}_m$, and since $m > 2D$, it follows that $S \cap (-S) = \{0\}$. Here, the decoding is done by the *full-residue* decoder, whose up to m terms annihilate every nonzero residue; Corollary 4.9 specifies how the interval structure of S motivates this choice, as well as the resulting parameter regime.

Setting parameters, take $M = \ell$ and $w = \Theta(\ell / \log \ell)$. Then $D = \mathsf{poly}(\ell)$, so $\log m = O(\log \ell)$, yielding catalytic space $O(d \log m) = O(\ell)$ bits. The gadget makes $O(m) = \mathsf{poly}(\ell)$ oracle calls and keeps $O(\log \ell)$ bits for bookkeeping, yielding an algorithm for $\mathsf{TreeEval}_{h,\ell}$ with space $O(\ell + h \log \ell) = O(\log n \log \log n)$ and running in time $\mathsf{poly}(\ell^h, 2^{\ell}) = n^{O(\log \log n)}$, recovering the result of [CM24] (made precise in Corollary 4.9). Note that here, the catalytic space is so small that we are simply regarding it as ordinary free space, so this algorithm does not need to rely on the catalytic space model. In the terms of Table 1, the low dimension $d = O(\ell / \log \ell)$ is why no catalytic tape is needed, while the decoder's $\mathsf{poly}(\ell)$ sparsity over a $\mathsf{poly}(\ell)$-size prime modulus costs $O(\log \ell)$ bits at each of the h levels and $\mathsf{poly}(\ell)$ recursive calls per node, which is where the super-logarithmic space and super-polynomial time come from.

Finally, we can also adapt our instantiation to recover the improvements of Stoeckl [Sto23] and Goldreich [Gol24] that achieved $O(\log n \log\log n/\log\log\log n)$ space. In the space bound above, the two costs are unbalanced: the registers cost $O(\ell)$ bits once, while the bookkeeping costs $O(\log \ell)$ bits at each of the h levels. The idea of [Sto23, Gol24] is to rebalance the two by increasing the branching factor of the $\mathsf{TreeEval}_{h,\ell}$ instance: group b consecutive levels of the tree into a single node of fan-in 2^b, so that the recursion depth drops to h/b. Our framework is very compatible with this tweak: instead of requiring $S + S$ to not wrap around, we require $\underbrace{S + \ldots + S}_{2^b \text{ terms}}$ to not wrap around. So we take the prime m to now be $O(2^b D)$ instead of just $O(D)$. As long as $2^b \leq \ell$, say, we still have $\log m = O(\log \ell)$, so we still only have $O(\log \ell)$ bits of bookkeeping space for each internal node, while we now need $O(2^b)$ catalytic registers of $O(\ell)$ bits each. The total space is thus $O(2^b \ell + (h/b)\log \ell)$, up to lower-order terms, and optimizing over b recovers the $O(\log n \log\log n/\log\log\log n)$ space bound of [Sto23, Gol24]. We make this precise in Section 4.5.

Thinking back to Remark 2.6, we emphasize that this last optimization is where the sumset decoding route becomes essential. Under the alternative route of multiplying two decoders, every child of an internal node would have its own independent monomial index, so an internal node would need $2^b \cdot O(\log \ell)$ bits of bookkeeping, as opposed to the $O(\log \ell)$ achieved with the sumset route. (Remark 4.13 makes this precise.)

As summarized in Table 1, better ingredients translate directly into better algorithms; Corollary 4.10 quantifies what shorter matching-vector families would give.

3 Preliminaries

Notation. For an integer N, we write $[N]$ to be the set $\{1, 2, \ldots, N\}$. We use $\mathsf{reg} \leftarrow \mathsf{state}$ to indicate that the register reg is updated to hold the contents of state. We let $\exp(\cdot)$ and $\log(\cdot)$ denote the base-two exponential and logarithm, though the choice of base will not affect our results.

3.1 The Catalytic-Space Model

We first define catalytic machines:

Definition 3.1. *A catalytic machine M is defined as a Turing machine in the usual sense—i.e., a read-only input tape, a write-only output tape, and a (space-bounded) read-write work tape—with an additional read-write tape known as the catalytic tape. Unlike the ordinary work tape, the catalytic tape is initialized to hold an arbitrary string τ, and M has the restriction that for any initial setting of the catalytic tape, at the end of its computation the catalytic tape must be returned to the original state τ.*

We parameterize such a catalytic computation by three resources: time, space, and catalytic space.

Definition 3.2. *Let $\mathsf{CatSpaceTime}[C(n), S(n), T(n)]$ be the class of languages recognized by catalytic machines that, on inputs of size $n \in \mathbb{N}$, use $O(S(n))$ workspace and $O(C(n))$ catalytic space and run in time $O(T(n))$ in the worst case.*

Remark 3.3 (Representations of rings on the catalytic tape). *Our algorithms (and those of Cook and Mertz and many others) interpret the catalytic tape as holding a vector in $\mathbb{Z}_m^{3d}$—the three registers of Theorem 4.4—though the definition of catalytic space gives a tape that consists of bits, with arbitrary initial contents that must be restored exactly. This translation requires care: for example, if we naively represent $\mathbb{Z}_3$ with 2 bits, the tape could hold values that do not correspond to an element of the ring. A standard trick (cf. [CP25]) handles this at*

the cost of $B := \lceil \log(3dm) \rceil + 1$ *bits per element and an additive* $\mathsf{poly}(dm)$ *term in the runtime. Partition the tape into* $3d$ *blocks of* B *bits and let* $K := \lfloor 2^B/m \rfloor$*. Call a block holding the integer* $z \in [0, 2^B)$ valid *if* $z < Km$*, and write a valid block as* $z = qm + r$ *with* $0 \le r < m$*: the block represents the ring element* r*, and an update* $r \leftarrow r + c \bmod m$ *leaves* q *untouched, so it keeps the block valid and is invertible. Each block has fewer than* m *invalid values, so fewer than* $3dm < 2^B$ *of the shifts* $\Delta \in [0, 2^B)$ *make some block* $(z + \Delta) \bmod 2^B$ *invalid; the algorithm finds a shift* Δ *making every block valid by exhaustive search in time* $\mathsf{poly}(dm)$*, applies it to every block, and stores* Δ *in* $B = O(\log(dm))$ *bits of workspace. It then runs with all register operations acting on the* r*-parts. Since the algorithm restores every register modulo* m*, each* r*-part returns to its initial value, and the frozen* q*-parts never change; subtracting* Δ *modulo* 2^B *from every block at the end therefore restores the tape to its initial bit contents exactly.*

3.2 Existing Algorithms for Tree Evaluation

We recall prior algorithms for tree evaluation. The lowest-space procedure for tree evaluation is a brilliant algorithm due to Cook and Mertz:

Theorem 3.4 ([CM24]). $\mathsf{TreeEval}_{h,\ell} \in \mathsf{Space}[O(\log(n) \cdot \log\log(n))]$.

A further result of Stoeckl [Sto23], written up by Goldreich [Gol24], reduces the space by a $\log\log\log(n)$ factor.

Furthermore, the one-hot algorithm of Cook and Mertz [CM20, Theorem 1] is a register program over $\mathbb{F}_2$ with three registers of 2^ℓ bits each that works from arbitrary initial register contents and restores them; run on a catalytic tape, it gives:

Theorem 3.5 ([CM20, Theorem 1]). $\mathsf{TreeEval}_{h,\ell} \in \mathsf{CatSpaceTime}\left[O(2^\ell), O(h + \ell), \mathsf{poly}(n)\right]$*; in particular,* $\mathsf{TreeEval} \in \mathsf{CatSpaceTime}\left[O(\sqrt{n}), O(\log n), \mathsf{poly}(n)\right]$.

Alternatively, the observation that $\mathsf{TreeEval}_{h,\ell}$ can be computed by an unbounded fan-in circuit of depth $O(h)$ and size $\mathsf{poly}(n)$, combined with the catalytic circuit-evaluation algorithm of [BCK+14], gives the weaker bound $\mathsf{TreeEval}_{h,\ell} \in \mathsf{CatSpaceTime}\left[n^c, O(\log n), n^c\right]$.

Finally, we can reduce from $\mathsf{TreeEval}$ with constant (but greater than 2) fan-in to $\mathsf{TreeEval}$ with fan-in 2, at a mild cost to the height. The instances of interest in Section 5 are exponentially large and presented implicitly, so we state the reduction at the level of single bits: it converts access to individual entries of the r-ary instance into access to individual entries of a binary instance.

Lemma 3.6. *Let* $r \ge 2$*. There is an algorithm that, given access to individual truth-table entries and leaf values of a* $\mathsf{TreeEval}_{h,\ell,r}$ *instance* I *(Definition 2.2), computes any requested bit of a* $\mathsf{TreeEval}_{h\lceil \log r \rceil, \ell \cdot \lceil r/2 \rceil, 2}$ *instance* I' *whose root value is the root value of* I *padded with zeros. It uses* $O(h \log r + \ell r)$ *bits of workspace beyond that access, and makes at most one access to* I *per requested bit.*

Proof. We set $\ell' = \ell \cdot \lceil r/2 \rceil$ and think of every value as an ℓ-bit string padded to length ℓ'. For every internal node u of I with children $u_1, \dots, u_r$, we place a complete binary gadget tree of height $\lceil \log r \rceil$ whose leaves are the gadgets of $u_1, \dots, u_r$ together with dummy slots, assigning the r original children to the gadget's leaves by splitting them as evenly as possible at every level (the first $\lceil a/2 \rceil$ of a children to the left subtree, the rest to the right). A dummy slot below a bottom-level node of I is a leaf of I' valued 0^ℓ; a dummy slot elsewhere is filled with a complete subtree of gates that ignore their children and output $0^{\ell'}$, over $0^{\ell'}$-valued leaves, so that I' is a complete binary tree of height $h\lceil \log r \rceil$. Every non-root node of the gadget collects the values passed from its children: a gadget node at distance $j \ge 1$ from the gadget root then

holds the concatenation of the (unpadded) ℓ-bit values of the at most $\lceil r/2^j \rceil \le \lceil r/2 \rceil$ original children below it—iterated halving, since $\lceil \lceil a/2 \rceil /2 \rceil = \lceil a/4 \rceil$—padded to length ℓ', which fits. (A naive left-to-right assignment would not fit: for $r = 5$, one child of the gadget root would hold four values.) The gadget root, whose collected inputs are $v_{u_1}, \ldots, v_{u_r}$, has output $f_u(v_{u_1}, \ldots, v_{u_r})$ padded by $0^{\ell'-\ell}$. This preserves the value at the root of I, up to the padding.

For the bit access: from the index of a requested bit of I', identify the original node u, the position within its gadget, and the position within an ℓ'-bit value, by integer arithmetic on the index. A collection gate's entry is a fixed selection of the bits of its two arguments, computed directly from the index; a gadget-root entry requires reading one truth-table entry of I, at an argument tuple extracted from the index; a constant-$0^{\ell'}$ gate's entry is computed from the index alone; and a leaf of I' is a leaf of I, padded, or $0^{\ell'}$. The index has length $O(h \log r + \ell r)$, and the arithmetic above needs no more workspace than that. □

3.3 Matching Vector Families

We next define matching-vector families and verify useful facts about them. Let $m > 1$ be a positive integer modulus.

Definition 3.7 (Log-space uniformity)**.** *Recall the definition of S-matching-vector families in Definition 2.4. We say such a (sequence of) families $\{\mathbf{u}_i^{(N)}, \mathbf{v}_i^{(N)} : N \in \mathbb{N}, i \in [N]\}$ is log-space uniform if there is an algorithm that, on input $(1^N, i, j, m)$ where $i, j \in [N]$, prints $(\mathbf{u}_i, \mathbf{v}_j)$ to the output tape and runs in space $O(\log N + \log d + \log m)$.*[3]

Remark 3.8 (Space-s uniformity and coordinate access)*. More generally, we say such a family is* space-s uniform *if the printing algorithm above runs in space $O(s)$; log-space uniformity is the case $s = \log N + \log d + \log m$. When a construction depends on auxiliary parameters (such as a list of prime factors of m, or the digit bound and width in Lemma 3.12), the printing algorithm receives them as part of its input; each construction below states its parameters. Two standard observations about this interface will be used without further comment. First, an algorithm that needs a single coordinate $(\mathbf{u}_i)_c$ or $(\mathbf{v}_j)_c$, rather than the whole vector, can run the printing algorithm, suppress all output before the cth coordinate, and stop after producing it; this costs $O(\log d)$ additional space for a counter. Second, a space-$O(s)$ algorithm runs in time $2^{O(s)}$ (assuming the input has length at most $2^{O(s)}$), which suffices for all of our running-time claims.*

We describe two well-known instantiations of S-matching-vector families, which will be a crucial ingredient in our construction and in our unified presentation with the Cook-Mertz algorithm.

Construction 1: An S-matching-vector family with a composite modulus m and restricted set S.

Definition 3.9 (Canonical set)**.** *Let $t \ge 1$ and suppose $p_1, \ldots, p_t$ are distinct primes and $m = p_1 p_2 \ldots p_t$. We say that the canonical set of $\mathbb{Z}_m$ is the set $S_m^* \subseteq \mathbb{Z}_m$ of residues that are either 0 or 1 mod p_i for all $i \in [t]$; in particular $0 \in S_m^*$. The 2^t patterns of residues give distinct elements by the Chinese remainder theorem, so $|S_m^*| = 2^t$.*

Theorem 3.10 ([BBR94, Gro00, DGY11])**.** *Let $m = p_1 p_2 \ldots p_t$ be a product of $t \ge 1$ distinct primes, given as input, let S be the canonical set of $\mathbb{Z}_m$, and let $H \ge w \ge 1$ be integers. Then, there exists a space-$O\big(\log N + (mw)^{1/t} \log(H+1) + \log m\big)$-uniform S-matching-vector family over $\mathbb{Z}_m^d$ of size N, where:*

$$N = \binom{H}{w} \qquad \text{and} \qquad d \le \sum_{j=0}^{\lfloor (mw)^{1/t} \rfloor} \binom{H}{j}.$$

[3] Note that we allow space logarithmic in the modulus (versus doubly logarithmic). This does not affect the ultimate complexity of any of our constructions.

In both parameter regimes of Corollary 3.11, the space bound is $O(\log N + \log m)$, so those families are log-space uniform; this is verified in Section C.

Proof sketch. This follows from setting parameters suitably in Lemma B.1, our account of [DGY11, Lemma 11]. For each $i \in [t]$, take e_i to be minimal such that $p_i^{e_i} > (mw)^{1/t}/p_i$ (possibly $e_i = 0$, when $p_i > (mw)^{1/t}$). This implies that $\prod_i p_i^{e_i} > mw/\prod_i p_i = w$, and moreover for any i we have $p_i^{e_i} \le (mw)^{1/t}$, so the dimension $\sum_{j<\max_i p_i^{e_i}} \binom{H}{j}$ given by Lemma B.1 is at most the bound displayed above, and the space bound $O(\log N + \delta \log H + \log m)$ of Lemma B.1 is at most the one displayed above since $\delta \le (mw)^{1/t}$ and $\log H \le \log(H+1)$. While [DGY11, Lemma 11] does not state any uniformity, it is immediate from their construction; we verify this in Section B. □

The following corollary is straightforward; we defer its proof to Section C.

Corollary 3.11. *All of the following hold:*

1. *([Gro00]) Let $t \ge 2$ be a constant, $p_1, \dots, p_t$ be t fixed odd primes, $m = p_1 p_2 \dots p_t$, and S be the canonical set of $\mathbb{Z}_m$. Then, for any $\ell \ge 1$, there exists a log-space-uniform S-matching-vector family over $\mathbb{Z}_m^d$ of size exactly 2^ℓ with dimension $d = \exp\left(O_m\left(\ell^{1/t}(\log(\ell+2))^{1-1/t}\right)\right)$, where the implied constant depends on the fixed primes.*

2. *For any ℓ larger than an absolute constant, let $t = \left\lceil \sqrt{\log \ell} \right\rceil$, $p_1, \dots, p_t$ be the first t odd primes, $m = p_1 p_2 \dots p_t$, and S be the canonical set of $\mathbb{Z}_m$. Then, there exists a log-space-uniform S-matching-vector family over $\mathbb{Z}_m^d$ of size exactly 2^ℓ with dimension $d = \exp(\exp(O(\sqrt{\log \ell})))$.*

Construction 2: An S-matching-vector family with an arbitrary, odd modulus m and a larger set S.

Lemma 3.12. *Let M, w be positive integers and let $S = \{0, 1, \dots, wM^2\}$. Let $m > wM^2$ be an odd prime, with M, w, and m given as input. Then there exists a space-$O(\log N + \log m)$ uniform S-matching-vector family over $\mathbb{Z}_m^{w+2}$ of size $N = M^w$.*

Proof. Index the elements of $[N]$ as vectors in $[M]^w$, identifying $i \in [N]$ with the base-M digit vector of $i-1$ (with digits shifted into $[M]$); both directions of this identification are computable in space $O(\log N)$ by standard arithmetic. For $x \in [M]^w$, define:

$$\mathbf{u}_x := \left(1, x_1, \dots, x_w, \sum_{c=1}^{w} x_c^2\right), \text{ and}$$
$$\mathbf{v}_x := \left(\sum_{c=1}^{w} x_c^2, -2x_1, \dots, -2x_w, 1\right).$$

For any $x, y \in [M]^w$, we have $\langle \mathbf{u}_x, \mathbf{v}_y \rangle = \sum_{c=1}^{w} (x_c - y_c)^2 \in [0, wM^2]$, where the containment holds over the integers. Since $m > wM^2$, it follows that the inner product is 0 mod m if and only if $x = y$, so this is an S-matching-vector family.

For uniformity: on input $(1^N, i, j, m)$, the algorithm recovers the digit vectors x of i and y of j and prints the $w+2$ coordinates of $\mathbf{u}_x$ and $\mathbf{v}_y$ in order. Each coordinate is 1, a digit $x_c \in [M]$, a value $-2y_c$ reduced mod m, or the sum $\sum_c x_c^2$, which is a nonnegative integer smaller than m and is computed by accumulation. All quantities involved are at most m^2, so the space used is $O(\log N + \log m)$. □

Remark 3.13. *The score set S above contains 0, as our convention requires, and the hypothesis $m > wM^2$ makes the diagonal inner products the only ones that vanish. The noncancellation property $S \cap (-S) = \{0\}$ required by our main theorem is stronger: it holds if and only if $m > 2wM^2$. The instantiations in Section 4 choose the prime m accordingly.*

The instantiations generate the prime m themselves, using the following standard fact.

Lemma 3.14 (Prime selection)**.** *Given a positive integer D in binary, the least prime in $(2D, 4D)$ exists and can be found in space $O(\log D)$.*

Proof. Existence follows from Bertrand's postulate (for every $n > 1$ there is a prime strictly between n and $2n$), applied with $n = 2D$. The algorithm tests the integers in $(2D, 4D)$ in increasing order by trial division; every quantity involved is at most $4D$, so the algorithm runs in space $O(\log D)$ (and time $\mathsf{poly}(D)$). □

3.4 A-Decoders

Our decoders are applied to sums of scores, so we first fix notation for sumsets.

Definition 3.15 (Sumsets and noncancellation)**.** *For $S \subseteq \mathbb{Z}_m$ and an integer $r \geq 1$, the* r-fold sumset *of S is $rS := \{s_1 + \cdots + s_r : s_1, \ldots, s_r \in S\}$; we write $S + S := 2S$. We say S is* r-noncancelling *if $s_1 + \cdots + s_r = 0$ with $s_1, \ldots, s_r \in S$ implies $s_1 = \cdots = s_r = 0$. When $0 \in S$, being 2-noncancelling is equivalent to $S \cap (-S) = \{0\}$.*

A second ingredient of our construction is a family of sparse A-decoders, which we defined in Definition 2.8 and is heavily inspired by the S-decoding polynomials defined by Efremenko [Efr09]. Our algorithm must evaluate decoder terms on demand, so we also fix an access interface and count its cost.

Definition 3.16 (Uniform A-decoder)**.** *Let $(\varphi_1, \ldots, \varphi_k; b_1, \ldots, b_k)$ be an A-decoder over $\mathbb{Z}_m$ (Definition 2.8), presented together with a parameter string; for the constructions below, the parameters are the prime factors $p_1, \ldots, p_t$ of m (Construction 1) or m itself (Construction 2). We say the decoder is* space-s uniform *if there is an algorithm that, given the parameters, an index $j \in [k]$, and $z \in \mathbb{Z}_m$, outputs the shift b_j and the value $\varphi_j(z)$ in space $O(s)$.*

Our A-decoders are distilled from the following polynomial formulation, a ring-valued counterpart of the field-valued Definition 2.5, which is how decoding is organized in the matching-vector-codes literature [Efr09, DGY11]. We call it *weak* because its second condition asks only that the constant coefficient of $P(1)$ be 1, rather than that $P(1) = 1$; over the ring $\mathcal{R}$ below, the stronger requirement cannot be met once A contains an element of prime-power additive order (Lemma 2.7), as every set A we use does. We record the formulation and then the exact correspondence.

Definition 3.17 (Weak A-decoding polynomials)**.** *Let $m > 1$ be a positive integer, let $A \subseteq \mathbb{Z}_m$ be a subset containing 0, and let $\mathcal{R} = \mathbb{Z}_m[X]/(X^m - 1)$. We say a polynomial $P \in \mathcal{R}[Z]$ is a* weak A-decoding polynomial *if:*

1. *for every $a \in A \setminus \{0\}$, we have $P(X^a) = 0$, and*
2. *$P(X^0) = P(1)$ has constant coefficient equal to 1. (The* constant coefficient *of an element of $\mathcal{R}$ is the coefficient of X^0 in its canonical representative $\sum_{u=0}^{m-1} c_u X^u$.)*

The sparsity of P is its number of non-zero coefficients (equivalently, its number of monomials).

The definition of A-decoders may look ad hoc next to the polynomial formulation above, but the two notions contain exactly the same information: an A-decoder is a weak decoding polynomial with its coefficients extracted. The following lemma makes this precise.

Lemma 3.18 (Equivalence of decoders and weak decoding polynomials)**.** *Let $m > 1$, let $A \subseteq \mathbb{Z}_m$ be a set containing 0, and let $\mathcal{R} = \mathbb{Z}_m[X]/(X^m - 1)$. For every $k \geq 1$, there exists an A-decoder over $\mathbb{Z}_m$ of sparsity at most k if and only if there exists a weak A-decoding polynomial over $\mathcal{R}$ of sparsity at most k. The translation preserves the shifts: the decoder $(\varphi_1, \dots, \varphi_k; b_1, \dots, b_k)$ and the polynomial $P(Z) = \sum_{j=1}^{k} C_j Z^{b_j}$ correspond via $C_j = \sum_{y \in \mathbb{Z}_m} \varphi_j(y) X^{-y}$, i.e., $\varphi_j(y)$ is the coefficient of X^{-y} in C_j.*

Proof. First suppose $(\varphi_1, \dots, \varphi_k; b_1, \dots, b_k)$ is an A-decoder, and define C_j and P as in the statement, merging any terms with equal shifts (which only lowers the sparsity). For any $a \in \mathbb{Z}_m$, substituting $y = z + ab_j$ in each term gives

$$P(X^a) = \sum_{j=1}^{k} \sum_{y \in \mathbb{Z}_m} \varphi_j(y)\, X^{ab_j - y} = \sum_{z \in \mathbb{Z}_m} \left(\sum_{j=1}^{k} \varphi_j(z + ab_j) \right) X^{-z}. \tag{5}$$

For $a \in A \setminus \{0\}$, every inner sum on the right vanishes by the first decoder condition, so $P(X^a) = 0$. For $a = 0$, the constant coefficient of $P(1)$ is the $z = 0$ inner sum, namely $\sum_j \varphi_j(0) = 1$ by the second condition. Hence P is a weak A-decoding polynomial of sparsity at most k.

Conversely, given a weak A-decoding polynomial $P(Z) = \sum_{j=1}^{k} C_j Z^{b_j}$ with distinct shifts b_j, write $C_j = \sum_{u \in \mathbb{Z}_m} c_{j,u} X^u$ and define $\varphi_j(y) := c_{j,-y}$, so that (5) again holds: the coefficient of X^{-z} in $P(X^a)$ is $\sum_j \varphi_j(z + ab_j)$. Then $P(X^a) = 0$ for all $a \in A \setminus \{0\}$ gives the first decoder condition for every $z \in \mathbb{Z}_m$, and the constant coefficient of $P(1)$ being 1 gives $\sum_j \varphi_j(0) = 1$. □

We next give two constructions of A-decoders, corresponding to the parameter regimes for the S-matching-vector family constructions given above. Although our definition of A-decoders is new (to the best of our knowledge), these decoders are assembled from standard ingredients (cf. [Efr09]).

Construction 1: An A-decoder for composite m and the sumset of the canonical set.

Lemma 3.19. *Let $p_1, \dots, p_t$ be t distinct odd primes, given as input, and let $m = p_1 p_2 \dots p_t$. Let $A = S_m^* + S_m^*$ (Definition 3.15); equivalently, A is the set of residues in $\mathbb{Z}_m$ that are in $\{0, 1, 2\} \bmod p_i$ for all $i \in [t]$. Then, there exists a space-$O(\log m)$ uniform A-decoder over $\mathbb{Z}_m$ with sparsity $\leq 3^t$, whose term-access algorithm moreover runs in time $\mathsf{poly}(\log m)$.*

Proof. We first check that the two descriptions of A agree. An element of $S_m^* + S_m^*$ is, modulo each p_i, a sum of two residues in $\{0, 1\}$, hence lies in $\{0, 1, 2\} \bmod p_i$. Conversely, given a in $\{0, 1, 2\} \bmod$ every p_i, decompose $a \bmod p_i$ as $0 + 0$, $1 + 0$, or $1 + 1$ for each i, and combine the first and the second components by the Chinese remainder theorem into $s, s' \in S_m^*$ with $s + s' = a$.

For each $i \in [t]$, define $\varphi_0^{(i)}, \varphi_1^{(i)}, \varphi_2^{(i)} : \mathbb{Z}_{p_i} \to \mathbb{Z}$ by:

$$\begin{aligned} \varphi_0^{(i)}(z) &= -\mathbb{1}[z = -2] \\ \varphi_1^{(i)}(z) &= \mathbb{1}[z = 0] + \mathbb{1}[z = -1] \\ \varphi_2^{(i)}(z) &= -\mathbb{1}[z = 1]. \end{aligned}$$

The relevant facts about $\varphi^{(i)}$ are the following:

- We have:
$$\sum_{c=0}^{2} \varphi_c^{(i)}(0) = 1. \tag{6}$$
Note that this relies on p_i being odd.

- For $a \in \{1, 2\}$, we have:
$$\sum_{c=0}^{2} \varphi_c^{(i)}(z + ac) = -\mathbb{1}[z = -2] + \mathbb{1}[z + a = 0] + \mathbb{1}[z + a = -1] - \mathbb{1}[z + 2a = 1] = 0. \tag{7}$$
Indeed, when $a = 1$, the above holds because the RHS is $-\mathbb{1}[z = -2] + \mathbb{1}[z = -1] + \mathbb{1}[z = -2] - \mathbb{1}[z = -1] = 0$, and when $a = 2$ it holds because the RHS is $-\mathbb{1}[z = -2] + \mathbb{1}[z = -2] + \mathbb{1}[z = -3] - \mathbb{1}[z = -3] = 0$.

Both facts hold as identities of integer-valued functions on $\mathbb{Z}_{p_i}$. With this in mind, we define our A-decoder as follows: we index the terms by elements $\beta = (\beta_1, \dots, \beta_t) \in \{0, 1, 2\}^t$, and define $b_\beta = \mathsf{CRT}(\beta_1, \dots, \beta_t)$ and $\varphi_\beta(z) := \prod_{i \in [t]} \varphi_{\beta_i}^{(i)}(z) \bmod m$. Note that this is well-defined as each factor is periodic modulo a divisor of m. Now, for any $z \in \mathbb{Z}_m$ and $a \in \mathbb{Z}_m$, we have, modulo m,

$$\begin{aligned}
\sum_{\beta \in \{0,1,2\}^t} \varphi_\beta(z + ab_\beta) &\equiv \sum_{\beta \in \{0,1,2\}^t} \prod_{i \in [t]} \varphi_{\beta_i}^{(i)}(z + ab_\beta) \\
&\equiv \sum_{\beta \in \{0,1,2\}^t} \prod_{i \in [t]} \varphi_{\beta_i}^{(i)}(z + a\beta_i) && (b_\beta \equiv \beta_i \bmod p_i) \\
&\equiv \prod_{i \in [t]} \left(\sum_{c \in \{0,1,2\}} \varphi_c^{(i)}(z + ac) \right). && \text{(distributivity)}
\end{aligned}$$

Now, if $a \in A \backslash \{0\}$, there exists an $i \in [t]$ such that $a \equiv 1$ or $2 \pmod{p_i}$ (if $a \equiv 0$ modulo every prime factor then $a = 0$), and by (7) that factor in the above product is zero for every z, over the integers and hence modulo m. On the other hand, if $z = a = 0$, the above simplifies to $\prod_{i \in [t]} \left(\sum_{c \in \{0,1,2\}} \varphi_c^{(i)}(0) \right) = \prod_{i \in [t]} 1 = 1$ by (6). This verifies both conditions of Definition 2.8, so this is an A-decoder of sparsity $\leq 3^t$.

Finally, uniformity. Identify the index set $[3^t]$ with $\{0, 1, 2\}^t$ by base-3 counting, in space $O(t)$. Given β and $z \in \mathbb{Z}_m$, the shift is computed as $b_\beta = \sum_{i \in [t]} \beta_i \cdot \frac{m}{p_i} \cdot \left(\left(\frac{m}{p_i} \right)^{-1} \bmod p_i \right) \bmod m$, where each inverse is obtained by the extended Euclidean algorithm; the value $\varphi_\beta(z)$ is computed by reducing z modulo each p_i, evaluating $\varphi_{\beta_i}^{(i)}$ by comparing the residue against 0, ± 1, and -2, and multiplying the t factors modulo m. Every intermediate quantity is at most m^2, so the space used is $O(\log m)$, and since $t \leq \log_3 m$, the total time is $\mathsf{poly}(\log m)$. □

Construction 2: A full-residue A-decoder for an odd prime m.

Lemma 3.20. *Let m be an odd prime, given as input, and let $A = \mathbb{Z}_m$. Then, there exists a space-$O(\log m)$ uniform A-decoder over $\mathbb{Z}_m$ with sparsity $\leq m$.*

Proof. Index the terms by $j \in \mathbb{Z}_m$, and define $b_j = j$ and $\varphi_j : \mathbb{Z}_m \to \mathbb{Z}_m$ by:
$$\varphi_j(z) = \mathbb{1}[z \neq 0] + \mathbb{1}[j = 0].$$

Now for any $a \in \mathbb{Z}_m$ and any $z \in \mathbb{Z}_m$, we have:

$$\sum_{j \in \mathbb{Z}_m} \varphi_j(z + aj) = 1 + \sum_{j \in \mathbb{Z}_m} \mathbb{1}[z + aj \neq 0]$$
$$= 1 + \left|\{j \in \mathbb{Z}_m \,:\, z + aj \neq 0\}\right|.$$

When $a \not\equiv 0 \pmod m$, the map $j \mapsto z + aj$ is a bijection on $\mathbb{Z}_m$ because m is prime, so the count is $m - 1$ and the sum is $m \equiv 0 \pmod m$. When $z = a = 0$, the sum is equal to 1. It follows that this is an A-decoder as desired. Uniformity is immediate: given j and z, the shift $b_j = j$ and the value $\varphi_j(z)$ are computed by two comparisons in space $O(\log m)$ (and time $\mathsf{poly}(\log m)$). □

4 Main Result: Catalytic Tree Evaluation from Matching Vectors

Throughout this section we work with the following objects, each presented by its parameters and uniform in the sense of Remark 3.8 and Definition 3.16:

- a modulus $m > 1$ and a score set $S \subseteq \mathbb{Z}_m$ with $0 \in S$ and $S \cap (-S) = \{0\}$, i.e., S is 2-noncancelling (Definition 3.15);
- an $(S + S)$-decoder $(\varphi_1, \dots, \varphi_k; b_1, \dots, b_k)$ over $\mathbb{Z}_m$ (Definitions 2.8 and 3.15);
- an S-matching-vector family $\{\mathbf{u}_x, \mathbf{v}_x \in \mathbb{Z}_m^d \,:\, x \in \{0,1\}^\ell\}$ of size 2^ℓ (Definition 2.4); and
- an auxiliary *output family* $\{\mathbf{w}_y \in \mathbb{Z}_m^d \,:\, y \in \{0,1\}^\ell\}$, needed only by Theorem 4.4; the proof of Theorem 4.1 provides its own output families, so instantiations need not provide one.

We write s for a common bound on the space of the access algorithms above, and assume $s \geq \log(dm)$. The noncancellation hypothesis $S \cap (-S) = \{0\}$ ensures that a sum of two scores vanishes only when both do, which is why the decoder is taken for the sumset $S + S$; recall from Lemma 3.19 that the composite construction provides exactly such a decoder for $S = S_m^*$. For the canonical set this hypothesis is automatic: if s' and $-s'$ both lie in S_m^* and m is odd, then s' is 0 or 1 and also 0 or -1 modulo each prime factor, hence $s' \equiv 0$ modulo each prime factor, i.e., $s' = 0$.

Our main theorem is stated at the level of ring-valued registers; two corollaries then translate it into the catalytic model and into ordinary space.

Theorem 4.1 (Main theorem, register form)**.** *In the setting above, there is an algorithm that is given, on its input tape, a* $\mathsf{TreeEval}_{h,\ell}$ *instance (the truth table of a function* $f_u : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$ *for every* $u \in \{0,1\}^{<h}$*, and a value* $v_u \in \{0,1\}^\ell$ *for every* $u \in \{0,1\}^h$*), and, as global state, three registers* $\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \boldsymbol{\tau} \in \mathbb{Z}_m^d$ *with arbitrary initial contents. The algorithm writes* $v_\emptyset$ *to its output tape and restores all three registers to their initial contents. It uses* $O(\ell + h(\log k + \log m) + s)$ *bits of workspace and runs in time* $(O(k))^h \cdot \mathsf{poly}(2^\ell, k, d, 2^s)$*.*

Corollary 4.2 (Catalytic form)**.** *In the setting above, we can solve* $\mathsf{TreeEval}_{h,\ell}$ *in* $O(\ell + h(\log k + \log m) + s)$ *free space,* $O(d \log(dm))$ *catalytic space, and* $(O(k))^h \cdot \mathsf{poly}(2^\ell, k, d, 2^s)$ *time.*

Proof. Interpret the catalytic tape as the three registers via Remark 3.3: this costs a catalytic tape of length $O(d \log(dm))$, an additional $O(\log(dm)) = O(s)$ bits of free space for the stored shift, and an additive $\mathsf{poly}(dm)$ term in the runtime. Run the algorithm of Theorem 4.1 on the represented registers; it restores them modulo m, which by Remark 3.3 restores the tape exactly. □

Corollary 4.3 (Ordinary-space form)**.** *In the setting above, we can solve* $\mathsf{TreeEval}_{h,\ell}$ *by an ordinary (non-catalytic) algorithm using* $O(d \log m + \ell + h(\log k + \log m) + s)$ *space and* $(O(k))^h \cdot \mathsf{poly}(2^\ell, k, d, 2^s)$ *time.*

Proof. Allocate the three registers in ordinary workspace, as $3d$ blocks of $\lceil \log m \rceil$ bits initialized to zero; every intermediate state is then a genuine element of $\mathbb{Z}_m^{3d}$, so no representation shift is needed, and Theorem 4.1 applies verbatim. In this form the final restoration pass of the algorithm may be skipped, though running it does not affect correctness; the restorations *during* the recursion remain essential to correctness. □

The instantiations of this framework—the two Grolmusz-based settings behind the introduction's headline results, and a squared-distance setting recovering the Cook–Mertz–Goldreich algorithm [CM24, Gol25]—are collected in Section 4.4; the extension of the framework to higher-arity trees, and the blocked bounds it yields, are in Section 4.5.

4.1 Recursive Step: One Level of Tree Evaluation

The main technical workhorse for our results is the following theorem. It is the formal counterpart of the informal one-level gadget $\mathsf{OneLevelGadget}$ of Definition 2.3: the two families $y \mapsto \mathbf{u}_y$ and $y \mapsto \mathbf{v}_y$ play the role of Enc on the input side, selected per query by the ctrl bit; the output family $\{\mathbf{w}_y\}$ plays it on the output side; and the interface spells out what the informal definition elides—positional register addressing, and exactly how much state is retained across a query.

Theorem 4.4 (One level)**.** *In the setting of this section, consider an algorithm given the following resources:*

- *global state comprising three registers* $\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \boldsymbol{\tau} \in \mathbb{Z}_m^d$*, held in this fixed order; the oracle below addresses the registers by their position, regardless of how the algorithm may have modified or permuted their contents;*
- *the truth table of a function* $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$ *on the input tape; and*
- *access to an oracle* $\mathcal{O}$ *that is parameterized by two labels* $x_0, x_1 \in \{0,1\}^\ell$*, which are fixed throughout the algorithm's execution and are not otherwise accessible to the algorithm. On query* $(\alpha, \mathsf{ctrl}, \sigma) \in \mathbb{Z}_m \times \{0,1\} \times \{0,1\}$*, given in local space, the oracle updates the register in position* σ *as*

$$\boldsymbol{\tau}_\sigma \leftarrow \begin{cases} \boldsymbol{\tau}_\sigma + \alpha\, \mathbf{u}_{x_\sigma} & \text{if } \mathsf{ctrl} = 0, \\ \boldsymbol{\tau}_\sigma + \alpha\, \mathbf{v}_{x_\sigma} & \text{if } \mathsf{ctrl} = 1, \end{cases}$$

  and leaves the other two registers unchanged.

Then there is an algorithm that, given $\alpha^* \in \mathbb{Z}_m$ *in local space, performs the net update*

$$\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} + \alpha^* \cdot \mathbf{w}_{f(x_0,x_1)}$$

and leaves $\boldsymbol{\tau}_0$ *and* $\boldsymbol{\tau}_1$ *unchanged, for* every *initial state of the three registers; moreover, every update it makes to* $\boldsymbol{\tau}_0$ *or* $\boldsymbol{\tau}_1$ *is later undone by the exactly inverse update. It makes* $4k + 4$ *oracle queries. Between queries it uses* $O(\ell + s + \log k)$ *bits of local space, of which only* $O(\log k + \log m)$ *bits are retained across any single query; and it runs in time* $\mathsf{poly}(2^\ell, k, d, 2^s)$*.*

Before we prove the theorem, we begin with a preliminary lemma; the underlying compute-and-compare use of inner products is standard in catalytic computing [BCK+14, CM24].

Lemma 4.5. *In the setting of Theorem 4.4, there is a procedure that computes the scalar*

$$g := \langle \boldsymbol{\tau}_0, \mathbf{v}_{x_0} \rangle + \langle \boldsymbol{\tau}_1, \mathbf{v}_{x_1} \rangle \in \mathbb{Z}_m,$$

where $\boldsymbol{\tau}_0, \boldsymbol{\tau}_1$ denote the register contents at the start of the procedure, and stores it in local space. It uses four oracle queries and $O(\log d + \log m)$ local space, of which $O(\log m)$ bits are retained across queries; it leaves all three registers unchanged, undoing each of its updates exactly.

Proof. The procedure is the following; recall that the oracle addresses registers by position.

1. Compute $\mathsf{tmp}_1 = \langle \boldsymbol{\tau}_0, \boldsymbol{\tau}_1 \rangle$ and write it into local space.
2. Swap the contents of the first two registers, coordinate by coordinate. By position, the state is now $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_0, \boldsymbol{\tau})$.
3. Query $\mathcal{O}(1, 1, 0)$: the oracle adds $\mathbf{v}_{x_0}$ to the register in position 0. The state is now $(\boldsymbol{\tau}_1 + \mathbf{v}_{x_0}, \boldsymbol{\tau}_0, \boldsymbol{\tau})$.
4. Compute $\mathsf{tmp}_2 = \langle \boldsymbol{\tau}_1 + \mathbf{v}_{x_0}, \boldsymbol{\tau}_0 \rangle$, the inner product of the first two registers, and write it into local space.
5. Query $\mathcal{O}(-1, 1, 0)$, restoring the state $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_0, \boldsymbol{\tau})$.
6. Query $\mathcal{O}(1, 1, 1)$: the state is now $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_0 + \mathbf{v}_{x_1}, \boldsymbol{\tau})$.
7. Compute $\mathsf{tmp}_3 = \langle \boldsymbol{\tau}_1, \boldsymbol{\tau}_0 + \mathbf{v}_{x_1} \rangle$ and write it into local space.
8. Query $\mathcal{O}(-1, 1, 1)$, and swap the contents of the first two registers back.

We have $\mathsf{tmp}_2 - \mathsf{tmp}_1 = \langle \mathbf{v}_{x_0}, \boldsymbol{\tau}_0 \rangle$ and $\mathsf{tmp}_3 - \mathsf{tmp}_1 = \langle \boldsymbol{\tau}_1, \mathbf{v}_{x_1} \rangle$, so the procedure outputs $g = (\mathsf{tmp}_2 - \mathsf{tmp}_1) + (\mathsf{tmp}_3 - \mathsf{tmp}_1)$. Each swap and each inner product of two registers is performed with a coordinate counter ($O(\log d)$ bits) and, for the inner products, an accumulator ($O(\log m)$ bits); across the oracle queries the procedure retains only the tmp values and its position in the list above, $O(\log m)$ bits in all. Every update made—the two swaps and the four oracle queries—is undone exactly. □

4.2 Proof of Theorem 4.4

We give the algorithm, then its correctness and its efficiency analysis. Except where stated, all arithmetic is carried out modulo m.

Algorithm. The input is a scalar $\alpha^* \in \mathbb{Z}_m$ in local space; the truth table of f is on the input tape.

1. Use Lemma 4.5 to compute and store $g = \langle \boldsymbol{\tau}_0, \mathbf{v}_{x_0} \rangle + \langle \boldsymbol{\tau}_1, \mathbf{v}_{x_1} \rangle$.
2. Repeat for each term index $j = 1, 2, \ldots, k$:
   (a) Obtain the shift b_j from the decoder's access algorithm, and make the two queries $\mathcal{O}(b_j, 0, 0)$ and $\mathcal{O}(b_j, 0, 1)$, i.e., the updates

$$\boldsymbol{\tau}_0 \leftarrow \boldsymbol{\tau}_0 + b_j \cdot \mathbf{u}_{x_0},$$
$$\boldsymbol{\tau}_1 \leftarrow \boldsymbol{\tau}_1 + b_j \cdot \mathbf{u}_{x_1}.$$

   (b) Repeat for all candidate labels $y_0, y_1 \in \{0, 1\}^\ell$:

i. Compute

$$c := \varphi_j\left(\langle \boldsymbol{\tau}_0, \mathbf{v}_{y_0}\rangle + \langle \boldsymbol{\tau}_1, \mathbf{v}_{y_1}\rangle - g\right) \in \mathbb{Z}_m.$$

This is a direct computation involving no oracle query: the candidates y_0, y_1 are explicit, so the coordinates of $\mathbf{v}_{y_0}$ and $\mathbf{v}_{y_1}$ are generated by the family's access algorithm, the inner products with the (readable) register contents are accumulated coordinate by coordinate, and φ_j is evaluated by the decoder's access algorithm.

ii. Read $f(y_0, y_1)$ from the truth table and apply, again directly and coordinate by coordinate, the update

$$\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} + \alpha^* \cdot c \cdot \mathbf{w}_{f(y_0,y_1)}.$$

(c) Make the two queries $\mathcal{O}(-b_j, 0, 0)$ and $\mathcal{O}(-b_j, 0, 1)$, restoring

$$\boldsymbol{\tau}_0 \leftarrow \boldsymbol{\tau}_0 - b_j \cdot \mathbf{u}_{x_0},$$
$$\boldsymbol{\tau}_1 \leftarrow \boldsymbol{\tau}_1 - b_j \cdot \mathbf{u}_{x_1}.$$

Lemma 4.6 (One-level correctness)**.** *For every initial state of the three registers and every $\alpha^* \in \mathbb{Z}_m$, the algorithm above performs the net update $\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} + \alpha^* \cdot \mathbf{w}_{f(x_0,x_1)}$ and leaves $\boldsymbol{\tau}_0, \boldsymbol{\tau}_1$ unchanged, undoing each of its updates to them exactly.*

Proof. The updates to $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$ come in exactly inverse pairs: steps (a) and (c) of iteration j add and subtract $b_j \mathbf{u}_{x_\sigma}$, and the updates inside Lemma 4.5 are undone there. So these registers end unchanged.

It remains to compute the net change to $\boldsymbol{\tau}$. During iteration j of step 2, the first two registers hold $\boldsymbol{\tau}_0 + b_j \mathbf{u}_{x_0}$ and $\boldsymbol{\tau}_1 + b_j \mathbf{u}_{x_1}$, where $\boldsymbol{\tau}_0, \boldsymbol{\tau}_1$ denote the initial contents, so the net change is

$$\sum_{j\in[k]} \sum_{y_0,y_1\in\{0,1\}^\ell} \alpha^* \cdot \varphi_j\left(\langle \boldsymbol{\tau}_0 + b_j\mathbf{u}_{x_0}, \mathbf{v}_{y_0}\rangle + \langle \boldsymbol{\tau}_1 + b_j\mathbf{u}_{x_1}, \mathbf{v}_{y_1}\rangle - g\right) \cdot \mathbf{w}_{f(y_0,y_1)}$$
$$= \alpha^* \cdot \sum_{y_0,y_1\in\{0,1\}^\ell} \mathbf{w}_{f(y_0,y_1)} \cdot \left(\sum_{j\in[k]} \varphi_j\left(\left(\langle \boldsymbol{\tau}_0, \mathbf{v}_{y_0}\rangle + \langle \boldsymbol{\tau}_1, \mathbf{v}_{y_1}\rangle - g\right) + b_j\left(\langle \mathbf{u}_{x_0}, \mathbf{v}_{y_0}\rangle + \langle \mathbf{u}_{x_1}, \mathbf{v}_{y_1}\rangle\right)\right)\right),$$

using linearity of the inner product.

We evaluate the inner sums. By the matching-vector property (Definition 2.4), each score $\langle \mathbf{u}_{x_\sigma}, \mathbf{v}_{y_\sigma}\rangle$ lies in S (here $0 \in S$ covers the diagonal pairs) and vanishes exactly when $y_\sigma = x_\sigma$. Hence $\langle \mathbf{u}_{x_0}, \mathbf{v}_{y_0}\rangle + \langle \mathbf{u}_{x_1}, \mathbf{v}_{y_1}\rangle \in S + S$, and since $S \cap (-S) = \{0\}$, this sum is 0 exactly when $(y_0, y_1) = (x_0, x_1)$. For $(y_0, y_1) \neq (x_0, x_1)$, then, it lies in $(S + S) \setminus \{0\}$, and the inner sum vanishes by the first condition of Definition 2.8, which holds for every value of its other argument. For $(y_0, y_1) = (x_0, x_1)$, the quantity $\langle \boldsymbol{\tau}_0, \mathbf{v}_{x_0}\rangle + \langle \boldsymbol{\tau}_1, \mathbf{v}_{x_1}\rangle - g$ is 0 by the definition of g, so the inner sum equals $\sum_j \varphi_j(0) = 1$ by the second condition. The net change to $\boldsymbol{\tau}$ is therefore $\alpha^* \cdot \mathbf{w}_{f(x_0,x_1)}$, as claimed. □

Efficiency analysis. The oracle queries are the four in step 1 and the four per iteration of step 2, i.e., $4k + 4$ in total. The bits retained across a query are: within step 1, the input α^* and the tmp values and list position ($O(\log m)$, by Lemma 4.5); within step 2, the term index j ($O(\log k)$), the scalars g, b_j, α^* ($O(\log m)$), and the current position in the loop body ($O(1)$)—in all, $O(\log k + \log m)$. Between queries, step 2(b) additionally uses the candidates y_0, y_1 (2ℓ bits), a coordinate counter ($O(\log d)$), accumulators and the scalar c ($O(\log m)$), a position on the input tape for the truth-table lookup ($O(\ell)$), and the workspace of the access algorithms ($O(s)$); since $s \geq \log(dm)$, the total is $O(\ell + s)$. For the runtime, step 2(b) dominates: it runs $k \cdot 2^{2\ell}$ times, and each run makes $O(d)$ coordinate steps, each involving access computations that run in time $2^{O(s)}$ (Remark 3.8); the total is $\mathsf{poly}(2^\ell, k, d, 2^s)$. □

4.3 Proof of Theorem 4.1

The recursion in this subsection is the one introduced by Cook and Mertz [CM24] (see also the exposition of Goldreich [Gol25]): evaluate the root by the one-level procedure, serving each of its oracle queries by recursing on the corresponding child. We prove it in full rather than citing it, because our resource bounds rest on details that differ from [CM24]: the oracle contract of Theorem 4.4 (positional registers, two hidden labels), the accounting of input access and of the state retained across queries, and the exact restoration that Corollary 4.2 extracts via Remark 3.3.

Throughout, nodes are indexed by $u \in \{0,1\}^{\le h}$ with the root $\emptyset$ at depth 0 and the leaves $u \in \{0,1\}^h$ at depth h, and v_u denotes the value of the $\mathsf{TreeEval}_{h,\ell}$ instance at node u. We assume without loss of generality that $d \ge E := \lceil \ell / \lfloor \log m \rfloor \rceil$, padding every vector of the matching-vector family with zero coordinates otherwise (this preserves all hypotheses—the access algorithms append the zero coordinates with a counter—and increases d by at most E). For $y \in \{0,1\}^\ell$, let $\mathsf{enc}(y) \in \mathbb{Z}_m^d$ denote the vector whose first E coordinates hold the $\lfloor \log m \rfloor$-bit blocks of y, each read as an element of $\mathbb{Z}_m$, and whose remaining coordinates are 0.

The recursion invokes Theorem 4.4 with three output families: $\{\mathbf{u}_y\}$ and $\{\mathbf{v}_y\}$, whose access algorithms are those of the matching-vector family, and $\{\mathsf{enc}(y)\}$, whose coordinates are computed directly from y in space $O(\ell + \log(dm)) = O(\ell + s)$.

Workspace allocation. The algorithm maintains:

- a node register holding the current node $u \in \{0,1\}^{\le h}$ and its depth ($O(h)$ bits);
- for each depth $0 \le i < h$, a slot of $O(\log k + \log m)$ bits holding the retained state of the depth-i call on the call stack, saved while that call waits for a recursive call to return (the calls on the stack lie along a single root-to-node path, so there is at most one per depth): its term index, scalars, phase, output-family selector, and pending query parameters;
- a single shared block of $O(\ell + h + s)$ bits for the scratch space of the currently active call (the h covers positions on the global input tape; see below); and
- a readout register reg of E elements of $\mathbb{Z}_m$ ($O(\ell + \log m)$ bits).

The sharing in the third item is the point of Theorem 4.4's guarantee that only $O(\log k + \log m)$ bits are retained across a query: a call uses its scratch space only between its own queries, and while a query is being served—which is when a child call runs—that call's scratch space is dead and the block can be reused by the child. The total is $O(\ell + h(\log k + \log m) + s)$ bits.

Input access. Theorem 4.4 speaks of an input tape holding a single truth table; the recursion gives the call at node u the table $f := f_u$ by address translation. The base offset of f_u's truth table within the $\mathsf{TreeEval}_{h,\ell}$ input is computed from u with an $O(h+\ell)$-bit counter, counted in the shared scratch block, and each access moves the input head across a tape of length $2^{O(h+\ell)}$; the runtime recurrence below accounts for these seeks. Leaf values $v_{u\sigma}$ are located in the same way.

Handling oracle queries. Suppose the one-level call at an internal node u makes the query $(\alpha, \mathsf{ctrl}, \sigma)$; the hidden labels of its oracle are the children's values $x_0 = v_{u0}$ and $x_1 = v_{u1}$. If $|u| = h-1$, the child $u\sigma$ is a leaf: the algorithm reads $v_{u\sigma}$ directly from the input tape and applies the update $\boldsymbol{\tau}_\sigma \leftarrow \boldsymbol{\tau}_\sigma + \alpha \cdot \mathbf{u}_{v_{u\sigma}}$ (or with $\mathbf{v}_{v_{u\sigma}}$ if $\mathsf{ctrl} = 1$) coordinate by coordinate, generating the vector with the family's access algorithm, in scratch space $O(\ell + h + s)$. Otherwise, the algorithm: saves the call's retained bits in the depth-$|u|$ slot; swaps

the contents of the registers in positions σ and 2 (coordinate by coordinate, with an $O(\log d + \log m)$-bit counter); updates the node register to $u\sigma$; and starts a fresh one-level call at $u\sigma$ with input $\alpha^* := \alpha$ and output family $\{\mathbf{u}_y\}$ if $\mathsf{ctrl} = 0$, $\{\mathbf{v}_y\}$ if $\mathsf{ctrl} = 1$. When that call returns, it swaps positions σ and 2 back, restores the node register to u, and resumes from the saved slot.

Correctness. We claim, by induction on the height of the subtree below u: a one-level call at an internal node u, run with input α^* and any of the three output families $\{\mathbf{w}_y\}$, changes the register in position 2 by $+\alpha^* \cdot \mathbf{w}_{v_u}$ and leaves the other two registers unchanged—for every state of the registers at the start of the call. Indeed, the oracle contract of Theorem 4.4 with hidden labels v_{u0}, v_{u1} is implemented faithfully: at $|u| = h - 1$ directly from the input tape, and at $|u| < h - 1$ by the child call, since after swapping positions σ and 2, the child (by the inductive hypothesis, with v_u replaced by $v_{u\sigma}$) adds $\alpha \cdot \mathbf{u}_{v_{u\sigma}}$ or $\alpha \cdot \mathbf{v}_{v_{u\sigma}}$ to position 2—which at that moment holds the contents of $\boldsymbol{\tau}_\sigma$—and leaves the other positions unchanged, so that after swapping back, the net effect is exactly the queried update, and note that $f_u(v_{u0}, v_{u1}) = v_u$. Given the oracle, Lemma 4.6 yields the claim at u.

Readout and restoration. At the top level, the algorithm first copies the first E coordinates of the register in position 2 into reg. It then makes the one-level call at the root with $\alpha^* = 1$ and output family $\{\mathsf{enc}(y)\}$; by the claim above, this adds $\mathsf{enc}(v_\emptyset)$ to position 2. The algorithm reads the first E coordinates of position 2 and, in local space, subtracts the stored reg values modulo m, decodes $v_\emptyset$ from the differences, and writes it to the output tape; the registers are not modified by this read. Finally it makes the call at the root once more, with $\alpha^* = -1$ and the same output family, which adds $-\mathsf{enc}(v_\emptyset)$ to position 2; all three registers now hold their initial contents, as Theorem 4.1 requires. (Note that the second run is correct regardless of the first: the claim above holds for every starting state of the registers.)

Runtime. For $1 \le \eta \le h$, let $T(\eta)$ bound the time of a one-level call at a node u with $\eta = h - |u|$. By Theorem 4.4, the call's own work is $2^{O(h)} \cdot \mathsf{poly}(2^\ell, k, d, 2^s)$, where the $2^{O(h)}$ factor accounts for the input-tape seeks of the address translation above, and each of its $4k + 4$ queries costs $2^{O(h)} \cdot \mathsf{poly}(2^\ell, d, 2^s)$ for a leaf update ($\eta = 1$) or two register swaps plus $T(\eta - 1)$ otherwise. Hence

$$T(\eta) \le (4k + 4) \cdot T(\eta - 1) + 2^{O(h)} \cdot \mathsf{poly}(2^\ell, k, d, 2^s), \qquad T(1) = 2^{O(h)} \cdot \mathsf{poly}(2^\ell, k, d, 2^s),$$

so $T(h) = (O(k))^h \cdot \mathsf{poly}(2^\ell, k, d, 2^s)$, the $2^{O(h)}$ factors being absorbed into $(O(k))^h$. The two root runs and the readout change this by at most a constant factor plus lower-order additive terms, giving the stated runtime. □

4.4 Instantiations

We instantiate the framework three ways. The first two use Grolmusz's matching vectors with the composite-modulus decoder and give the catalytic results of the introduction; the third uses the squared-distance vectors with the full-residue decoder and recovers the guarantees of Cook and Mertz [CM24] within the same framework, with no catalytic tape. In each case we specify the matching-vector family, the decoder, the modulus, and the access-space bound s, and read off the resources from Corollary 4.2 or Corollary 4.3.

Corollary 4.7. *For any $\epsilon > 0$, we can solve $\mathsf{TreeEval}_{h,\ell}$ in $O(\ell + h)$ free space, $\exp(O(\ell^\epsilon))$ catalytic space, and $\mathsf{poly}(2^{\ell+h})$ time.*

Proof. Let $t = \max(2, \lceil 3/\epsilon \rceil)$, let $p_1, \ldots, p_t$ be the first t odd primes, and let $m = p_1 \cdots p_t$ and $S = S^*_m$, so that m and the decoder sparsity $k \le 3^t$ (Lemma 3.19) are constants. Instantiate Corollary 4.2 with

the matching-vector family of Item 1 of Corollary 3.11, which has size exactly 2^ℓ and dimension $d = \exp\left(O_m\left(\ell^{1/t}(\log(\ell+2))^{1-1/t}\right)\right) \le \exp(O(\ell^\epsilon))$. All access algorithms run in space $O(\ell+\log d+\log m) = O(\ell)$, so we may take $s = O(\ell)$. The resources are then $O(\ell+h)$ free space, $O(d\log(dm)) \le \exp(O(\ell^\epsilon))$ catalytic space, and $2^{O(h)} \cdot \mathsf{poly}(2^\ell) = \mathsf{poly}(2^{\ell+h})$ time. □

Corollary 4.8. *Suppose ℓ is larger than a sufficiently large absolute constant. Then we can solve $\mathsf{TreeEval}_{h,\ell}$ in $O(\ell + h\sqrt{\log \ell}\log\log \ell)$ free space, $\exp(\exp(O(\sqrt{\log \ell})))$ catalytic space, and $\mathsf{poly}(2^{\ell+h\sqrt{\log \ell}})$ time.*

Proof. Let $t = \lceil\sqrt{\log \ell}\rceil$ and let $p_1, \dots, p_t$ be the first t odd primes. Then $\log m = O(t\log t)$, which is $O(\sqrt{\log \ell}\log\log \ell)$, and the decoder of Lemma 3.19 has sparsity $k \le 3^t = 2^{O(\sqrt{\log \ell})}$. Instantiate Corollary 4.2 with the family of Item 2 of Corollary 3.11, which has $d = \exp(\exp(O(\sqrt{\log \ell})))$; note $\log d = 2^{O(\sqrt{\log \ell})} = O(\ell)$ for ℓ large, so all access algorithms run in space $O(\ell)$ and we may take $s = O(\ell)$. The free space is then $O(\ell + h(\log k + \log m) + s) = O(\ell + h\sqrt{\log \ell}\log\log \ell)$. The catalytic space is $O(d\log(dm)) = \exp(\exp(O(\sqrt{\log \ell})))$. The time is $2^{O(h\sqrt{\log \ell})} \cdot \mathsf{poly}(2^\ell, d) = \mathsf{poly}(2^{\ell+h\sqrt{\log \ell}})$. (For ℓ below the constant, Corollary 4.7 with $\epsilon = 1$ already gives $O(h)$ free space and $O(1)$ catalytic space.) □

Corollary 4.9 (Cook–Mertz–Goldreich [CM24, Gol25])**.** *We can solve $\mathsf{TreeEval}_{h,\ell}$ by an ordinary (non-catalytic) algorithm using $O(\ell + h\log(\ell+2))$ space and $2^{O(\ell+h\log(\ell+2))}$ time. In particular, $\mathsf{TreeEval}$ is solvable in $O(\log n\log\log n)$ space and $n^{O(\log\log n)}$ time, the bounds of [CM24].*

Proof. Set the digit range $M := \max(2, \ell)$ and the number of digits $w := \lceil \ell/\lfloor\log M\rfloor\rceil$, so that $M^w \ge 2^\ell$, and let $D := wM^2$, which bounds the squared-distance scores. Using Lemma 3.14, generate the least prime $m \in (2D, 4D)$; since $m > 2D \ge 8$, the prime m is odd and exceeds wM^2, as Lemmas 3.12 and 3.20 require. All of M, w, D, m are computed from ℓ once, at the start, in space $O(\log D)$ and time $\mathsf{poly}(D)$. We instantiate the setting of this section as follows.

- Scores: $S := \{0, 1, \dots, D\}$, which contains 0; since $m > 2D$, we have $S \cap (-S) = \{0\}$ (Remark 3.13).
- Family: the squared-distance family of Lemma 3.12 over $\mathbb{Z}_m^{w+2}$, of size $M^w \ge 2^\ell$, restricted to its first 2^ℓ indices; a restriction changes neither the matching property nor the uniformity. Thus $d = w + 2$.
- Decoder: the full-residue decoder of Lemma 3.20, of sparsity $k \le m$. Its first defining condition holds for every nonzero residue, so in particular for every element of $(S + S) \setminus \{0\}$, and since the second condition does not involve the set A, it is therefore an $(S + S)$-decoder. (The score set certified by Lemma 3.12 is the whole interval $\{0, \dots, D\}$, so the sumset is all of $\{0, \dots, 2D\}$; this is why we use the full-residue decoder here.)
- Access space: the family's algorithm runs in space $O(\log(M^w) + \log m) = O(\ell + \log(\ell+2))$ and the decoder's in $O(\log m)$, so we may take $s = O(\ell + \log(\ell+2)) = O(\ell)$, and $s \ge \log(dm)$ holds.

Now apply Corollary 4.3. We have $d\log m = O\left(\left(\frac{\ell}{\log(\ell+2)} + 1\right)\log(\ell+2)\right) = O(\ell + \log(\ell+2))$, since $\lfloor\log M\rfloor = \Theta(\log(\ell+2))$, and $\log k + \log m = O(\log(\ell+2))$, so the space is

$$O(d\log m + \ell + h(\log k + \log m) + s) = O(\ell + h\log(\ell+2)). \quad (h \ge 1 \text{ absorbs the lone } \log(\ell+2) \text{ terms})$$

The time is

$$(O(k))^h \cdot \mathsf{poly}(2^\ell, k, d, 2^s) = 2^{O(h\log(\ell+2))} \cdot 2^{O(\ell)} = 2^{O(\ell+h\log(\ell+2))}.$$

For the final sentence, recall that $h, \ell \le O(\log n)$, so $\log(\ell+2) = O(\log\log n)$: the space is $O(\log n\log\log n)$ and the time is $2^{O(\log n\log\log n)} = n^{O(\log\log n)}$. □

Better tree evaluation from better ingredients. Improvements to the framework's two combinatorial ingredients enter the bounds only through the parameters d, m, k, and s of Corollaries 4.2 and 4.3. The most consequential case is the composite regime, where the dimension d is the bottleneck.

Corollary 4.10 (Tree evaluation from optimal matching vectors)**.** *Fix $m = p_1 \cdots p_t$, a product of $t \geq 2$ distinct odd primes, and suppose that there is a space-$O(\ell)$-uniform sequence of S_m^*-matching-vector families whose ℓth member has size 2^ℓ and lies in $\mathbb{Z}_m^d$ with $d = O(\ell \log(\ell + 2))$. Then* TreeEval *is solvable in $O(\log n \log\log n)$ ordinary space and polynomial time.*

Proof. Apply Corollary 4.3 with this family and the decoder of Lemma 3.19, whose sparsity $k \leq 3^t$ is constant, and take $s = O(\ell)$. The space is $O(d \log m + \ell + h(\log k + \log m) + s) = O(\ell \log(\ell + 2) + h)$ and the time is $(O(k))^h \cdot \mathsf{poly}(2^\ell, k, d, 2^s) = 2^{O(h+\ell)}$. Since $h, \ell \leq O(\log n)$, these are $O(\log n \log\log n)$ and $\mathsf{poly}(n)$. □

This would match the space bound of [CM24] with a polynomial running time, and with no catalytic tape at all. The corollary's dimension hypothesis matches, up to a constant factor, the following lower bound. Bhowmick, Dvir, and Lovett [BDL14] showed that, under the polynomial Freiman–Ruzsa conjecture [Ruz99] over $\mathbb{Z}_m^d$ with m constant, every matching-vector family in $\mathbb{Z}_m^d$ has size $2^{O(d/\log d)}$, so a family of size 2^ℓ requires $d \geq \Omega(\ell \log \ell)$; the conjecture has since been proven by Gowers, Green, Manners, and Tao—over $\mathbb{F}_2^d$ in [GGMT25], and in the bounded-torsion generality needed here in [GGMT24]—so this bound is unconditional. Conversely, to the best of our knowledge, no known lower bound [TB98, BDL14, ADL+25] rules out families meeting the hypothesis, whereas the best known constructions (Corollary 3.11) have $d = \exp\left(O(\ell^{1/t}(\log(\ell + 2))^{1-1/t})\right)$.

4.5 Higher Arity and Blocking

The framework extends from binary to r-ary trees with only local changes, and the extension buys an improvement in space: grouping levels of a binary tree into one higher-arity level rebalances the one-time cost of the registers against the per-level cost of the bookkeeping, an idea of Stoeckl [Sto23] and Goldreich [Gol24]. We state the extension, then relate it to the blocked bounds. We use the r-fold sumsets rS and r-noncancellation of Definition 3.15.

Theorem 4.11 (Higher arity, register form)**.** *Let $r \geq 2$, and modify the setting of this section as follows: the score set S is r-noncancelling (Definition 3.15; for $r = 2$ this is $S \cap (-S) = \{0\}$); the decoder is an rS-decoder over $\mathbb{Z}_m$; and the global state comprises $r + 1$ registers $\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_{r-1}, \boldsymbol{\tau} \in \mathbb{Z}_m^d$. Then there is an algorithm that is given, on its input tape, a $\mathsf{TreeEval}_{h,\ell,r}$ instance, writes the root value to its output tape, and restores all $r + 1$ registers to their initial contents. It uses $O(r\ell + h(\log k + \log m + \log r) + s)$ bits of workspace and runs in time $2^{O(h \log r)} \cdot (O(k))^h \cdot \mathsf{poly}(2^{r\ell}, k, d, 2^s)$.*

Proof sketch. The binary proof carries over with the following changes; $r = 2$ recovers Theorems 4.1 and 4.4 verbatim.

One level. The oracle of Theorem 4.4 is now parameterized by r hidden labels $x_0, \ldots, x_{r-1}$, and a query $(\alpha, \mathsf{ctrl}, \sigma)$ with $\sigma \in \{0, \ldots, r-1\}$ updates the register in position σ. The algorithm first computes $g := \sum_i \langle \boldsymbol{\tau}_i, \mathbf{v}_{x_i} \rangle$ by the swap trick of Lemma 4.5, pairing the register in position i with the one in position $i + 1 \bmod r$ (two queries per child, $2r$ in all). Then, for each term index $j \in [k]$—a single index, shared by all r children—it adds $b_j \mathbf{u}_{x_i}$ to every child register (r queries), loops over all candidate tuples $(y_0, \ldots, y_{r-1}) \in (\{0,1\}^\ell)^r$ performing the direct update

$$\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} + \alpha^* \cdot \varphi_j\Big(\sum\nolimits_i \langle \boldsymbol{\tau}_i, \mathbf{v}_{y_i} \rangle - g \Big) \cdot \mathbf{w}_{f(y_0,\ldots,y_{r-1})},$$

and undoes the r additions, for $2r(k+1)$ queries in all. In the telescoping computation of Lemma 4.6, the score sum $\sum_i \langle \mathbf{u}_{x_i}, \mathbf{v}_{y_i} \rangle$ lies in rS, and it is 0 exactly when every $y_i = x_i$, by the matching property and r-noncancellation; off the diagonal the inner sum vanishes by the first condition of the rS-decoder, and on the diagonal g cancels the correction and the inner sum is 1. The bits retained across a query are $O(\log k + \log m + \log r)$ (term index, scalars, phase, and the pending query's σ); the scratch space between queries is $O(r\ell + s)$ (the candidate tuple, counters, accumulators, and the truth-table position); the call's own time is $\mathsf{poly}(2^{r\ell}, k, d, 2^s)$.

Recursion. As in Section 4.3: the node register costs $O(h \log r)$ bits, each depth gets a slot of $O(\log k + \log m + \log r)$ bits, the shared scratch block is $O(r\ell + h \log r + s)$ bits (the input tape now has length $2^{O(h \log r + r\ell)}$, with the same address translation), and a query $(\alpha, \mathsf{ctrl}, \sigma)$ at an internal node is served by swapping positions σ and r and recursing on child σ. The readout via enc and the two root runs are unchanged. The workspace totals as stated, and the recurrence $T(\eta) \le 2r(k+1)\, T(\eta - 1) + 2^{O(h \log r)} \cdot \mathsf{poly}(2^{r\ell}, k, d, 2^s)$ solves to the stated time: the coefficient contributes $(2r(k+1))^h = 2^{O(h \log r)} \cdot (O(k))^h$, and the input-tape seek factors contribute another $2^{O(h \log r)}$—which, unlike the binary case's $2^{O(h)}$, cannot be absorbed into the base of the power, so it stays explicit. Finally, the translations of Corollaries 4.2 and 4.3 apply with $r+1$ registers in place of three (the argument of Remark 3.3 is unchanged with $(r+1)d$ blocks); in particular, allocating the registers in initialized ordinary workspace costs $O(rd \log m)$ bits of additional space, the form used below. □

Theorem 4.12 (Stoeckl [Sto23]; [Gol24, Theorem 3])**.** *For every $1 \le b \le h$, $\mathsf{TreeEval}_{h,\ell}$ can be solved in space $O\big(h + 2^b \ell + \frac{h}{b} \log(\ell + 2)\big)$.*

The proof groups each run of b consecutive levels into a single node of fan-in 2^b, whose gate is evaluated on demand from the original binary gates, and applies the bound $O(r\ell + h \log(r\ell))$ for fan-in-r trees [CM24, Theorem 7] (see also [Gol24, Theorem 2]) with $r = 2^b$. Theorem 4.11 recovers that fan-in-r bound within our framework whenever $r \le \mathsf{poly}(\ell)$: instantiate it as in Corollary 4.9, but with the prime m chosen in $(rD, 2rD)$, so that S is r-noncancelling: a sum of r scores cannot vanish modulo m unless every score does. The registers then cost $O(r\ell)$ bits, and each level of the recursion retains $O(\log(r\ell))$ bits. Taking $b = \lfloor 0.9 \log\log \ell \rfloor$ in Theorem 4.12 gives the space bound $O(\log n \log\log n / \log\log\log n)$ [Sto23, Gol24]; since this block size has $2^b \le \ell$, our framework recovers this bound as well.

Remark 4.13 (Sum versus product). *Note that Theorem 4.11 decodes the* sum *of the r children's scores; this is what the rS-decoder and r-noncancellation are for. Applying an S-decoder to each child's masked score separately and multiplying the r results, as the one-hot algorithm of [CM20] does for $r = 2$, would avoid both, and also isolate the diagonal term. But the product of r sums expands into k^r terms, indexed by tuples $(j_1, \dots, j_r) \in [k]^r$ of* independent *term indices, one per child; as written, the recursion retains $r \log k$ bits per level (the current tuple) and makes $\Theta(rk^r)$ oracle queries per call, against $O(\log k + \log m + \log r)$ bits and $2r(k+1)$ queries for the sumset route. For $r = 2$ the difference is a constant factor in the space bound and in the exponent of the time bound, but for blocking it matters: with $r = 2^b$, grouping b levels divides the number of levels by b while multiplying the per-level bookkeeping by 2^b, erasing the savings of Theorem 4.12. The shared term index of the sumset route avoids this, so reducing the tree to h/b levels does save space. (We compare against this natural implementation of the product route, but it is conceivable some other implementation could avoid these overheads.)*

5 Application: New Time-Space-Catalytic Space Tradeoffs

We present two applications of our tree-evaluation algorithms: time-space tradeoffs in the catalytic model via the reduction of Williams [Wil25], and analogous tradeoffs for circuit evaluation via the reduction of Shalunov [Sha25].

5.1 Composing with Space-Bounded Reductions

The reductions in this section produce tree-evaluation instances of size exponential in the resources of interest, so no algorithm can afford to write one down. Instead, an instance I is *implicitly presented in space* s_P: a procedure P, given the index of a bit of I, outputs that bit using $O(s_P)$ bits of workspace and $2^{O(s_P)}$ time. Our algorithms run on such instances by the standard method of emulative composition [Gol08, Lemma 5.2] (see also [PT25, Proposition 3.3]): whenever the algorithm reads a bit of its input, run P to compute that bit. This adds $O(s_P)$ bits of workspace and multiplies the running time by $2^{O(s_P)}$; the index of the bit read is held by the algorithm itself, whose space bounds already account for input addressing (Section 4.3). Since P runs in ordinary workspace and never touches the catalytic tape, the catalytic guarantees of our algorithms are unaffected, and the root value they compute may be kept in an $O(\ell)$-bit buffer instead of being written to the output tape.

5.2 The Reduction of Williams

We now state the reduction of Williams [Wil25] from time-bounded computation to tree evaluation. Note that the simulation of [Wil25] does *not* reduce a computation to a single tree-evaluation instance, since we cannot compute the pattern of tape head movements in advance. Instead it produces a sequence of TEP instances, where each candidate pattern σ yields an (implicitly represented) instance that either computes the answer or declares failure at the root. Since t need not be time-constructible, he also enumerates candidate time bounds T. His construction gives the following.

Theorem 5.1 ([Wil25, Section 3.2])**.** *Let M be a multitape Turing machine deciding a language L in time $t(n) \geq n$, and let $b(T)$ be computable from T in space $O(\log T)$. There are constants $r = r(M) \geq 2$ and $c = c(M)$ such that the following holds on every input $x \in \{0,1\}^n$. For every $T \geq n$ with $\log T \leq b(T) \leq T$ and every $\sigma \in \{0,1\}^{\leq cT/b(T)}$, there is a $\mathsf{TreeEval}_{h,\ell,r}$ instance $I_{x,T,\sigma}$, with $h := \lceil cT/b(T) \rceil$ and $\ell := \lceil c\, b(T) \rceil$, implicitly presented in space $O(T/b(T) + b(T))$ given (x, T, σ), whose root value is either a designated string $\perp$, signaling failure (in particular whenever the simulated computation has not halted within T steps), or else contains a designated* verdict bit*, such that:*

1. *if $T \geq t(n)$, then for at least one σ the root value of $I_{x,T,\sigma}$ is not $\perp$; and*
2. *whenever the root value of $I_{x,T,\sigma}$ is not $\perp$, its verdict bit equals $L(x)$.*

Combining the reduction with our catalytic algorithms yields the tradeoffs. The first recovers the parameters announced in the introduction.

Corollary 5.2. *Let L be decided by a multitape Turing machine in time $t(n) \geq n$. For every $\epsilon > 0$,*

$$L \in \mathsf{CatSpaceTime}\left[2^{O(t^\epsilon)}, O(\sqrt{t}), 2^{O(\sqrt{t})}\right],$$

which is Corollary 1.3.

Proof. Apply Theorem 5.1 with $b(T) := \lceil \sqrt{T} \rceil$, which satisfies $\log T \le b(T) \le T$ for $T \ge 16$; the finitely many inputs with $n < 16$ are decided trivially. The algorithm enumerates $T = n, n+1, \ldots$ and, for each T, the strings $\sigma \in \{0,1\}^{\le cT/b(T)}$ in lexicographic order, holding T and σ in $O(\log T + T/b(T)) = O(\sqrt{T})$ bits of free space. For each pair, the instance $I_{x,T,\sigma}$ has $h, \ell = O(\sqrt{T})$ and constant fan-in r, and is implicitly presented in space $O(\sqrt{T})$; composing its presentation with Lemma 3.6 implicitly presents, still in space $O(\sqrt{T})$, a binary $\mathsf{TreeEval}_{h',\ell'}$ instance with $h', \ell' = O(\sqrt{T})$ whose root value is that of $I_{x,T,\sigma}$ padded with zeros. Run the algorithm of Corollary 4.7, with its parameter set to $\min(1, 2\epsilon)$, on this instance by emulative composition, capturing the root value in a workspace buffer; its first ℓ bits are the root value of $I_{x,T,\sigma}$ and the rest are zeros, by Lemma 3.6. If those ℓ bits are $\perp$, move on to the next pair; otherwise output the verdict bit and halt. By property 1 of Theorem 5.1, the algorithm halts by the time T reaches $t(n)$, and by property 2 its output is $L(x)$.

Resources: every T considered is at most $t = t(n)$. The free space is $O(\sqrt{t})$ for T, σ, the presentations, and the parameter computations, plus $O(\ell' + h') = O(\sqrt{t})$ from Corollary 4.7. The catalytic space is $\exp(O(\ell'^{\min(1,2\epsilon)})) = 2^{O(t^\epsilon)}$; the tape is restored at the end of each run, so a single tape serves every iteration and ends in its initial state. The time is at most $t \cdot 2^{O(\sqrt{t})} = 2^{O(\sqrt{t})}$ iterations, each costing $\mathsf{poly}(2^{\ell'+h'}) \cdot 2^{O(\sqrt{t})} = 2^{O(\sqrt{t})}$ by emulative composition. □

Optimizing the block length against the free space of Corollary 4.8 instead gives a sharper catalytic-space bound, at the cost of requiring more free space (albeit still less than that of [Wil25]).

Corollary 5.3. *Let L be decided by a multitape Turing machine in time $t(n) \ge n$. Then*

$$L \in \mathsf{CatSpaceTime}\left[\exp(\exp(O(\sqrt{\log t}))), O\big(\sqrt{t}\,(\log t)^{1/4}\sqrt{\log\log t}\,\big), 2^{O(\sqrt{t}\,(\log t)^{1/4}\sqrt{\log\log t})}\right].$$

Proof. Repeat the proof of Corollary 5.2 with $b(T) := \lceil \sqrt{T}\,(\log T)^{1/4}(\log\log T)^{1/2} \rceil$ and Corollary 4.8 in place of Corollary 4.7; the guard of Corollary 4.8 holds because $\ell' = \Theta(b(T))$ exceeds any absolute constant once n is large enough, and smaller inputs are decided trivially. Fix a guess $T \le t$ and write $b := b(T)$. Since $\log \ell' = \Theta(\log T)$ and $\log\log \ell' = \Theta(\log\log T)$, the free space is

$$\begin{aligned}
&O\big(T/b + b\big) + O\big(\ell' + h'\sqrt{\log \ell'}\,\log\log \ell'\big) \\
&\quad = O\big(b + (T/b)\sqrt{\log T}\,\log\log T\big) && (\ell' = \Theta(b),\ h' = O(T/b),\ T/b \le b) \\
&\quad = O\big(\sqrt{T}\,(\log T)^{1/4}\sqrt{\log\log T}\,\big), && \text{(the two terms balance at the chosen } b)
\end{aligned}$$

which is at most the displayed bound since $T \le t$. The catalytic space is $\exp(\exp(O(\sqrt{\log \ell'}))) = \exp(\exp(O(\sqrt{\log t})))$. For each T there are $2^{O(T/b)}$ iterations, each costing $\mathsf{poly}(2^{\ell'+h'\sqrt{\log \ell'}}) \cdot 2^{O(b)}$; since $h'\sqrt{\log \ell'} = O((T/b)\sqrt{\log T}) = O(b/\log\log T)$, each iteration is $2^{O(b)}$, and summing over the at most t guesses gives the displayed time. □

5.3 Circuit Evaluation

Shalunov [Sha25] subsequently gave a direct reduction from circuit evaluation to tree evaluation (and in this case, the reduction produces a single TEP instance that is guaranteed to correctly encode the circuit, so no enumeration is required).

Theorem 5.4 ([Sha25, Theorem 4.1 and Lemma 2.3])**.** *There is a constant c such that the following holds. Let C be a circuit with S gates (described by $O(S \log S)$ bits), let $x \in \{0,1\}^{\le S}$ be an input to C, and let $b \in [2, S]$.*

Then there is a binary $\mathsf{TreeEval}_{h,\ell,2}$ *instance with* $h \le c\,S/b$ *and* $\ell = b$*, implicitly presented from* (C, x) *in space* $O(b + \log^2 S)$*, a designated bit of whose root value equals* $C(x)$*. (The reduction of [Sha25, Theorem 4.1] presents the instance in space* $O(b + \log S)$ *from a topologically sorted circuit with its input substituted in; the* $O(\log^2 S)$ *accounts for bringing an arbitrary pair* (C, x) *into that form, per [Sha25, Lemma 2.3].)*

Corollary 5.5. *For any* $\epsilon > 0$*, size-*S *circuit evaluation can be decided in* $O(\sqrt{S})$ *free space,* $2^{O(S^\epsilon)}$ *catalytic space, and* $2^{O(\sqrt{S})}$ *time.*

Proof. Apply Theorem 5.4 with $b := \lceil \sqrt{S} \rceil$ and run the algorithm of Corollary 4.7, with parameter $\min(1, 2\epsilon)$, on the implicitly presented instance by emulative composition; output the designated bit of the root value. The free space is $O(b + \log^2 S) + O(\ell + h) = O(\sqrt{S})$; the catalytic space is $\exp(O(\ell^{\min(1,2\epsilon)})) = 2^{O(S^\epsilon)}$; the time is $\mathsf{poly}(2^{\ell+h}) \cdot 2^{O(\sqrt{S})} = 2^{O(\sqrt{S})}$. □

6 Statement on AI Use

Activity	**AI used**
Asking the research question	No
Coming up with the approach	No
Proof development	Yes
Writing and exposition	Yes
Checking for bugs	Yes
Other supporting tasks	Yes

Details (elaborating on the points above). The ideation and writing of the first version of this paper took place without the use of any AI tools. We later had the idea of using weak A-decoding polynomials (Definition 3.17) to make our presentation cleaner and more modular, and used Google Gemini to work out some of the details. Later, we asked GPT-6 Astra to simplify this even further, and it suggested the notion of A-decoders (Definition 2.8). Finally, to incorporate these changes and add exposition explaining our techniques, we used GPT-6 Astra and Claude (Opus 5, Opus 5.5, and Fable 5). This included drafting some proofs. We also used these models for proofreading.

7 Acknowledgements

AH and SR thank Vinod Vaikuntanathan for encouraging this work and for helpful discussions. We also thank James Cook, Ian Mertz, Gregory Meyer, and Ryan Williams for useful discussions, and anonymous reviewers for helpful feedback on earlier versions of this paper.

References

[ADL+25] Divesh Aggarwal, Pranjal Dutta, Zeyong Li, Maciej Obremski, and Sidhant Saraogi. Improved lower bounds for 3-query matching vector codes. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*, volume 325 of *LIPIcs*, pages 2:1–2:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. 28

[AM25] Aryan Agarwala and Ian Mertz. Bipartite matching is in catalytic logspace. *CoRR*, abs/2504.09991, 2025. 3

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. 3

[BBR94] David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4(4):367–382, December 1994. 4, 11, 16, 37

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992. 3

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *STOC*, 2014. 1, 3, 15, 22

[BDL14] Abhishek Bhowmick, Zeev Dvir, and Shachar Lovett. New bounds for matching vector families. *SIAM J. Comput.*, 43(5):1654–1683, 2014. 3, 28

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2000. 4, 7

[CFL+13] Yeow Meng Chee, Tao Feng, San Ling, Huaxiong Wang, and Liang Feng Zhang. Query-efficient locally decodable codes of subexponential length. *Computational Complexity*, 22(1):159–189, 2013. 4

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50. IEEE Computer Society, 1995. 7, 10

[CGKS98] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *J. ACM*, 1998. 7

[CM20] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 752–760. ACM, 2020. 1, 2, 3, 5, 10, 15, 29

[CM21] James Cook and Ian Mertz. Encodings and the tree evaluation problem. *Electron. Colloquium Comput. Complex.*, TR21-054, 2021. 2, 3

[CM22] James Cook and Ian Mertz. Trading time and space in catalytic branching programs. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. 3

[CM24] James Cook and Ian Mertz. Tree evaluation is in space $O(\log n \cdot \log\log n)$. In *STOC*, 2024. 1, 2, 3, 5, 8, 9, 13, 15, 22, 25, 26, 27, 28, 29

[CMW+12] Stephen Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Transactions on Computation Theory (TOCT)*, 3(2), 2012. 1, 3, 5

[CP25] James Cook and Edward Pyne. Efficient catalytic graph algorithms. *CoRR*, abs/2509.06209, 2025. 3, 14

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server PIR with subpolynomial communication. *J. ACM*, 2016. 2, 4, 10

[DGY11] Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM Journal on Computing*, 40(4):1154–1178, 2011. 2, 4, 12, 16, 17, 18, 36, 37, 38

[Efr09] Klim Efremenko. 3-query locally decodable codes of subexponential length. In *STOC*, 2009. 4, 10, 11, 12, 18, 19

[Efr12] Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 2012. 4

[EMP18] Jeff Edmonds, Venkatesh Medabalimi, and Toniann Pitassi. Hardness of function composition for semantic read once branching programs. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 15:1–15:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 3

[GGMT24] W. Timothy Gowers, Ben Green, Freddie Manners, and Terence Tao. Marton's conjecture in abelian groups with bounded torsion. *arXiv preprint arXiv:2404.02244*, 2024. 3, 28

[GGMT25] William Timothy Gowers, Ben Green, Freddie Manners, and Terence Tao. On a conjecture of Marton. *Annals of Mathematics*, 201(2):515–549, 2025. 28

[GKS25] Fatemeh Ghasemi, Swastik Kopparty, and Madhu Sudan. Improved PIR schemes using matching vectors and derivatives. In *STOC*, 2025. 2, 4, 10

[GLM+25] Ashrujit Ghoshal, Baitian Li, Yaohua Ma, Chenxin Dai, and Elaine Shi. Scalable multi-server private information retrieval. In *TCC*. Springer, 2025. 7

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. 30

[Gol24] Oded Goldreich. Solving tree evaluation in $o(\log n \cdot \log \log n)$ space. *Electron. Colloquium Comput. Complex.*, TR24-124, 2024. 2, 12, 13, 14, 15, 28, 29

[Gol25] Oded Goldreich. On the Cook-Mertz tree evaluation procedure. In Oded Goldreich, editor, *Computational Complexity and Local Algorithms - On the Interplay Between Randomness and Computation*, volume 15700 of *Lecture Notes in Computer Science*, pages 102–112. Springer, 2025. 4, 6, 22, 25, 27

[Gop06] Parikshit Gopalan. *Computing with Polynomials over Composites*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2006. 37

[GR26] Aparna Gupte and Seyoon Ragavan. Exponentially fewer-server PIR from sparser S-decoding polynomials, 2026. arXiv:2607.22033. 4, 36, 39

[Gro00] Vince Grolmusz. Superpolynomial size set-systems with restricted intersections mod 6 and explicit Ramsey graphs. *Comb.*, 20(1):71–86, 2000. 2, 4, 11, 16, 17

[HPV77] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM (JACM)*, 24(2), 1977. 3

[HR26] Alexandra Henzinger and Seyoon Ragavan. Two-server private information retrieval in sublinear time and quasilinear space. In *EUROCRYPT*, 2026. 4, 7

[HU68] John E Hopcroft and Jeffrey D Ullman. Relations between time and tape complexities. *Journal of the ACM (JACM)*, 15(3):414–427, 1968. 3

[IN19] Kazuo Iwama and Atsuki Nagao. Read-once branching programs for tree evaluation problems. *ACM Trans. Comput. Theory*, 11(1):5:1–5:12, 2019. 3

[IS10] Toshiya Itoh and Yasuhiro Suzuki. Improved constructions for query-efficient locally decodable codes of subexponential length. *IEICE Transactions on Information and Systems*, 93(2), 2010. 4

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, pages 364–373. IEEE, 1997. 7

[KY09] Kiran S Kedlaya and Sergey Yekhanin. Locally decodable codes from nice subsets of finite fields and prime factors of Mersenne numbers. *SIAM Journal on Computing*, 38(5):1952–1969, 2009. 4

[LBA25] Or Lasri, Amos Beimel, and Bar Alon. Simplified PIR and CDS protocols and improved linear secret-sharing schemes. In *TCC*. Springer, 2025. 4

[Liu13] David Liu. Pebbling arguments for tree evaluation. *CoRR*, abs/1311.0293, 2013. 3

[Ope26] OpenAI. Simulating One-Tape Time in Two-Fifths-Power Space. OpenAI Math Release preprint, September 2026. 3, 4

[PT25] Edward Pyne and Roei Tell. Composing low-space algorithms. *Electron. Colloquium Comput. Complex.*, TR25-140, 2025. 30

[Ruz99] Imre Ruzsa. An analog of Freiman's theorem in groups. *Astérisque*, 258(199):323–326, 1999. 28

[Sha25] Yakov Shalunov. Improved bounds on the space complexity of circuit evaluation. *Electron. Colloquium Comput. Complex.*, TR25-078, 2025. 30, 31, 32

[Sto23] Manuel Stoeckl. Private communication, 2023. 2, 12, 13, 14, 15, 28, 29

[TB98] Gábor Tardos and David A. Mix Barrington. A lower bound on the mod 6 degree of the Or function. *Comput. Complex.*, 7(2):99–108, 1998. 28

[Wil25] R. Ryan Williams. Simulating time with square-root space. In *STOC*, 2025. 1, 3, 5, 30, 31

[WY05] David Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *CCC*. IEEE, 2005. 7

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 2008. 4

[Yek12] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3), 2012. 4

A Impossibility of Decoding Polynomials in Characteristic m

In this appendix, we prove Lemma 2.7.

Proof of Lemma 2.7. Let $j^* \in A$ be an element whose additive order in $\mathbb{Z}_m$ is p^a, for a prime p and an integer $a \geq 1$; note that $j^* \neq 0$, since the order of j^* exceeds 1, and that $p \mid m$, since the order of any element of $\mathbb{Z}_m$ divides m.

Suppose toward a contradiction that such γ and P exist. Since j^* has additive order p^a, we have $j^* p^a \equiv 0 \pmod{m}$ and hence $\gamma^{j^* p^a} = 1$. Hence, working over $\mathcal{R}$, we have:

$$\begin{aligned}
\frac{m}{p} \cdot (\gamma^{j^*} - 1)^{p^a} &= \frac{m}{p} \cdot (\gamma^{j^* p^a} - 1) && \text{(since } p \mid \binom{p^a}{i} \text{ for } i \in (0, p^a)\text{; characteristic } m) \\
&= 0 && (\gamma^{j^* p^a} = 1) \\
\Rightarrow \frac{m}{p} \cdot (P(\gamma^{j^*}) - P(1))^{p^a} &= 0 && (X - 1 \text{ divides } P(X) - P(1)) \\
\Rightarrow \frac{m}{p} \cdot (0 - 1)^{p^a} &= 0 && \text{(since } j^* \in A \setminus \{0\}\text{)},
\end{aligned}$$

which contradicts the requirement that $\mathcal{R}$ have characteristic exactly m. □

B Verifying the Uniformity of the Matching-Vector Family

The matching-vector family of [DGY11, Lemma 11] is explicit, but no space bound for generating it is stated there. In this appendix we reproduce the construction and verify that it is log-space uniform, making no claims to originality.

Our notation differs from that of [DGY11], whose symbols h, d, n, and k already have other global meanings in this paper. We write H for the ground-set size (their h), w for the subset size, $q_i = p_i^{e_i}$ for the prime-power moduli, $\delta = \max_i q_i$ for the degree parameter (their d), N for the family size (their k), and d for the dimension of the resulting family (their n), matching the body's conventions for N and d. Recall from Definition 3.9 that the canonical set S_m^* contains 0; the canonical set of [DGY11] excludes 0, and accordingly the cross inner products below lie in the zero-free part $S' := S_m^* \setminus \{0\}$.

Lemma B.1. *Let $m = \prod_{i=1}^{t} p_i$ be a product of distinct primes that are given as input. Let $w \geq 1$ and $e_1, \ldots, e_t \geq 0$ be integers such that $\prod_{i=1}^{t} p_i^{e_i} > w$; write $q_i = p_i^{e_i}$ and $\delta = \max_i q_i$. Let $H \geq w$. Then there exists a space $O(\log N + \delta \log H + \log m)$-uniform[4] S_m^*-matching-vector family of size $N = \binom{H}{w}$ in $\mathbb{Z}_m^d$, where $d = \sum_{j=0}^{\delta-1} \binom{H}{j}$.*

Remark B.2. *As literally stated, [DGY11, Lemma 11] assumes $q_i > w^{1/t}$ for every i; inspecting the proof shows that the hypothesis $\prod_i q_i > w$ suffices and that exponents $e_i = 0$ are allowed [GR26, Footnote 2]. Since the full*

[4] In both regimes in which we invoke this lemma we have $\log N \geq \delta \log H$, verified in Section C, so this meets the definition of log-space uniformity.

proof is reproduced below, we state the relaxed form. Our dimension count $\sum_{j=0}^{\delta-1}\binom{H}{j}$ *is also marginally smaller than the* $\sum_{j=0}^{\delta}\binom{H}{j}$ *stated there, because the polynomials below have degree at most* $\delta-1$.

We first define polynomial matching families. The combinatorial content of the definition is [DGY11, Definition 35]; the uniformity clause is ours, and it demands enumeration of both the polynomials and the evaluation points, because the conversion in Lemma B.4 evaluates monomials at the points.

Definition B.3 (Polynomial matching family; see [DGY11, Definition 35])**.** *Let* $S_0 \subseteq \mathbb{Z}_m \setminus \{0\}$*. We say that a set of polynomials* $\mathcal{F} = \{f_1, \dots, f_N\} \subseteq \mathbb{Z}_m[z_1, \dots, z_H]$ *and a set of points* $\mathcal{X} = \{x_1, \dots, x_N\} \subseteq \mathbb{Z}_m^H$ *form a space-s uniform polynomial* S_0*-matching family of size* N *if*

- *for all* $i \in [N]$, $f_i(x_i) = 0$*;*
- *for all* $i, j \in [N]$ *such that* $i \neq j$, $f_j(x_i) \in S_0$*;*
- *there is a space-*$O(s)$ *algorithm that, given* $i \in [N]$*, prints the polynomial* f_i *(as a list of monomials with their coefficients) and prints the point* x_i*.*

Such a family can be turned into matching vectors by an observation of Sudan. For $i \in [N]$, let $\mathsf{supp}(f_i)$ denote the set of monomials in the support of the polynomial f_i, and define $\mathsf{supp}(\mathcal{F}) = \bigcup_{i=1}^{N} \mathsf{supp}(f_i)$.

Lemma B.4 ([DGY11, Lemma 36])**.** *Let* $\mathcal{F}, \mathcal{X}$ *be a space-s uniform polynomial* S_0*-matching family of size* N *over* $\mathbb{Z}_m$*, and let* $\mathcal{M} \supseteq \mathsf{supp}(\mathcal{F})$ *be a set of monomials that can be enumerated in space* $O(s)$*. Then there is a space-*$O(s + \log m)$ *uniform* $(S_0 \cup \{0\})$*-matching-vector family of size* N *in* $\mathbb{Z}_m^d$*, where* $d = |\mathcal{M}|$*. In particular, for* $S_0 = S' = S_m^* \setminus \{0\}$ *this is an* S_m^**-matching-vector family.*

Proof. Let $\textsc{mon}_1, \dots, \textsc{mon}_d$ be the monomials of $\mathcal{M}$ in enumeration order, and write

$$f_i = \sum_{l=1}^{d} c_{il} \cdot \textsc{mon}_l$$

with $c_{il} \in \mathbb{Z}_m$ (setting $c_{il} = 0$ for monomials outside $\mathsf{supp}(f_i)$). Let $\mathbf{u}_i \in \mathbb{Z}_m^d$ be the coefficient vector $(c_{i1}, \dots, c_{id})$, and let $\mathbf{v}_j \in \mathbb{Z}_m^d$ be the evaluation vector $(\textsc{mon}_1(x_j), \dots, \textsc{mon}_d(x_j))$. Then $\langle \mathbf{u}_i, \mathbf{v}_j \rangle = f_i(x_j)$, which is 0 when $i = j$ and lies in S_0 when $i \neq j$; so this is an $(S_0 \cup \{0\})$-matching-vector family.

For uniformity: to print $\mathbf{u}_i$, enumerate $\mathcal{M}$ and, for each monomial, print its coefficient in f_i by running the printing algorithm for f_i and matching monomials, in space $O(s)$. To print $\mathbf{v}_j$, enumerate $\mathcal{M}$ and evaluate each monomial at x_j: the printing algorithm provides the coordinates of x_j, and a monomial is evaluated by multiplying the values of its variables one at a time modulo m, in space $O(s + \log m)$. □

We also require low-degree polynomials that detect the Hamming weight modulo a prime power. The statement below is a reformulation of [DGY11, Lemma 37], stated as Theorem 2.16 in Gopalan's thesis [Gop06], with its origins in the work of Barrington, Beigel, and Rudich [BBR94].

Lemma B.5 (cf. [DGY11, Lemma 37]; [Gop06, Theorem 2.16])**.** *Let* p *be a prime, let* $e \geq 0$ *be an integer, and write* $q = p^e$*. Let* $w \geq 0$ *and* $H \geq 1$ *be integers. There is a symmetric multilinear polynomial* $f(z_1, \dots, z_H) \in \mathbb{Z}_p[z_1, \dots, z_H]$ *with* $\deg(f) \leq q - 1$ *such that for all* $x \in \{0,1\}^H$*:*

$$f(x) \equiv \begin{cases} 0 \pmod{p} & \text{if } \sum_j x_j \equiv w \pmod{q}, \\ 1 \pmod{p} & \text{otherwise,} \end{cases}$$

where $\sum_j x_j$ *is computed over* $\mathbb{Z}$*. Moreover,* f *can be printed in space* $O(q \log(H+1) + \log p)$*.*[5]

[5]The explicitness is not stated in the cited sources, but is immediate from the construction; we outline it in the proof.

Proof outline. For $e = 0$ we have $q = 1$: every weight is congruent to w modulo 1, and $f = 0$ works. Assume $e \geq 1$. A symmetric multilinear polynomial of degree $\leq q-1$ has the form $f = \sum_{j=0}^{q-1} c_j E_j(z)$, where E_j denotes the elementary symmetric polynomial of degree j (we reserve the letter e for the exponents). On an input $x \in \{0,1\}^H$ of Hamming weight σ we have $E_j(x) = \binom{\sigma}{j}$, so $f(x) = \sum_{j<q} c_j \binom{\sigma}{j}$. By Lucas's theorem, for $j < q$ the value $\binom{\sigma}{j} \bmod p$ depends only on the first e base-p digits of σ, i.e., $\binom{\sigma}{j} \equiv \binom{\sigma \bmod q}{j} \pmod p$. The matrix $\left(\binom{\sigma}{j} \bmod p\right)_{\sigma,j \in \{0,\ldots,q-1\}}$ is lower triangular with unit diagonal, hence invertible over $\mathbb{Z}_p$, so there is a choice of coefficients $(c_j)_{j<q}$ with $\sum_{j<q} c_j \binom{\sigma}{j} \equiv v(\sigma)$ for all $\sigma \in \{0,\ldots,q-1\}$, where $v(\sigma) := \mathbb{1}[\sigma \not\equiv w \pmod q]$; by the periodicity just noted, $f(x) \equiv v(\sigma \bmod q)$ then holds for every weight $\sigma \leq H$. Inverting the binomial transform gives the coefficients explicitly as $c_j = \sum_{\sigma=0}^{j} (-1)^{j-\sigma} \binom{j}{\sigma} v(\sigma) \bmod p$, computable in space $O(\log q + \log p)$ using Lucas's theorem for the binomial coefficients. To print f, enumerate the multilinear monomials of degree $< q$ over $z_1, \ldots, z_H$ in space $O(q \log H)$; the coefficient of each monomial depends only on its degree j and equals c_j. □

Combining the per-prime polynomials yields weight detection modulo m:

Corollary B.6 (cf. [DGY11, Corollary 38])**.** *With $p_1, \ldots, p_t$, m, w, q_i, and δ as in Lemma B.1, there is a space-$O(\delta \log H + \log m)$ algorithm that prints a multilinear polynomial $f(z_1, \ldots, z_H) \in \mathbb{Z}_m[z_1, \ldots, z_H]$ with $\deg(f) \leq \delta - 1$ such that for all $x \in \{0,1\}^H$ with $\sum_j x_j \leq w$:*

$$f(x) \bmod m \in \begin{cases} \{0\} & \text{if } \sum_j x_j = w, \\ S_m^* \setminus \{0\} & \text{if } \sum_j x_j < w. \end{cases}$$

In the above, $\sum_j x_j$ is computed over $\mathbb{Z}$.

Proof. For each $i \in [t]$, let $f_i \in \mathbb{Z}_{p_i}[z_1, \ldots, z_H]$ be the polynomial of Lemma B.5 for the prime power q_i and target weight w, and let f be the multilinear polynomial over $\mathbb{Z}_m$ obtained by combining the f_i coefficientwise via the Chinese remainder theorem, so that $f \equiv f_i \pmod{p_i}$ for every i and $\deg f \leq \max_i(q_i - 1) \leq \delta - 1$. Fix x of Hamming weight $\sigma \leq w$. Each $f_i(x)$ is 0 or 1 mod p_i, so $f(x) \bmod m \in S_m^*$. If $\sigma = w$ then every $f_i(x) \equiv 0 \pmod{p_i}$, so $f(x) \equiv 0 \pmod m$. If $\sigma < w$, then since $0 < w - \sigma \leq w < \prod_i q_i$ and the q_i are pairwise coprime, there is an i with $\sigma \not\equiv w \pmod{q_i}$, whence $f_i(x) \equiv 1 \pmod{p_i}$ and $f(x) \not\equiv 0 \pmod m$. For the space bound: each coefficient of f is obtained from the t per-prime coefficients by the Chinese remainder theorem in space $O(\log m)$, and the monomial enumeration costs $O(\delta \log H)$ as in Lemma B.5. □

Claim B.7. *Let $N = \binom{H}{w}$. There is a space $O(\log N + \log H)$-computable bijection from $[N]$ to sets $T \subseteq [H]$ of size w.*

Proof. It clearly suffices to build a bijection from $[0, N-1]$ to subsets $T \subseteq [0, H-1]$ of size w. For a given combination T, denote the elements of T in decreasing order as $c_w > \ldots > c_1 \geq 0$. We express this combination as the number

$$K = \binom{c_w}{w} + \binom{c_{w-1}}{w-1} + \ldots + \binom{c_1}{1}.$$

There is a greedy algorithm that prints $c_w, \ldots, c_1$ given K that runs in space $O(\log N + \log H)$ as follows. We let $S = 0$ and $i = w$ and choose c_i maximal such that

$$\binom{c_i}{i} \leq K - S \text{ and set } S \leftarrow S + \binom{c_i}{i}.$$

We can store S using $O(\log N)$ bits, since all its values are bounded by N, and each candidate c_i using $O(\log H)$ bits, so the algorithm runs in space $O(\log N + \log H)$. □

Proof of Lemma B.1. We construct a polynomial matching family and apply Lemma B.4.

We work with subsets $T \subseteq [H]$ of size w, indexed by $[N]$ via the space-$O(\log N + \log H)$ bijection of Claim B.7. Let f be the polynomial of Corollary B.6. For each such T, define f_T as the polynomial obtained from f by setting $z_j = 0$ for $j \notin T$, and let $x_T \in \{0,1\}^H$ be the indicator vector of T.

This is a polynomial S'-matching family: for size-w subsets T, L we have $f_T(x_L) = f(x_{T\cap L})$, and the weight $|T \cap L|$ is at most w, with equality if and only if $T = L$. By Corollary B.6, $f_T(x_T) = 0$ and $f_T(x_L) \in S' = S_m^* \setminus \{0\}$ for $T \neq L$.

For the uniformity clause: given $i \in [N]$, the algorithm computes T via Claim B.7 and prints x_T directly; it prints f_T by enumerating the multilinear monomials of degree $< \delta$ and printing, for each, the coefficient given by Corollary B.6 if all of the monomial's variables lie in T, and 0 otherwise. This takes space $O(\log N + \delta \log H + \log m)$.

Finally, apply Lemma B.4 with $\mathcal{M}$ the set of all multilinear monomials of degree $< \delta$ over $z_1, \dots, z_H$, which can be enumerated in space $O(\delta \log H)$. This yields a space-$O(\log N + \delta \log H + \log m)$ uniform S_m^*-matching-vector family of size N and dimension $|\mathcal{M}| = \sum_{j=0}^{\delta-1} \binom{H}{j}$, as claimed. □

C Proof of Corollary 3.11

The parameter calculation in this appendix is essentially that of [GR26, Corollary 3.4], and we claim no novelty for it: we apply Theorem 3.10 with the parameter choices made in the proof of that corollary and quote its computation of the resulting dimension. What we verify in addition is what the present paper needs beyond existence: the uniformity side condition promised by the footnote of Lemma B.1 and the final claim of Theorem 3.10, the exact family size, and how the prime factors are given.

We first set up the dictionary with [GR26]. Their database size n is our 2^ℓ, so their $\log n$ is our ℓ; their number of primes, which they call ℓ, is our t; and their parameter is $\eta = \ell^{t-1}/m$. The proof of [GR26, Corollary 3.4], in its second case, which applies once $\log \eta \geq \Omega(t^2)$, instantiates the construction with

$$\theta = \Theta\left(\frac{\log \eta}{t^2}\right), \qquad w \sim \frac{\ell}{\theta}, \qquad H \sim \frac{\ell}{\theta} \cdot 2^\theta,$$

so that $N = \binom{H}{w} \geq (H/w)^w = 2^\ell$ and $(mw)^{1/t} \sim \ell/(\eta\theta)^{1/t}$, and computes

$$\sum_{j=0}^{\lfloor (mw)^{1/t} \rfloor} \binom{H}{j} \leq \exp\left(O\left(\frac{\ell (\log \eta)^{1-1/t}}{t\, \eta^{1/t}}\right)\right). \tag{8}$$

Concretely, we take $\theta = \max\big(10, \lceil \log \eta / t^2 \rceil\big)$, $w = \lceil \ell/\theta \rceil$, and $H = w \cdot 2^\theta$, all computable from the input in space $O(\ell + \log m)$; then $N = \binom{H}{w} \geq (H/w)^w = 2^{\theta w} \geq 2^\ell$ and $\log N \leq w(\theta + \log e) = O(\ell)$, and the facts quoted above hold for this choice. By Theorem 3.10 with these choices of H and w, we obtain a space-$O(\log N + (mw)^{1/t} \log(H+1) + \log m)$-uniform S_m^*-matching-vector family of size $N \geq 2^\ell$ and dimension bounded by (8). Restricting the family to its first 2^ℓ indices under the indexing of Claim B.7 gives a family of size exactly 2^ℓ; a restriction changes neither the matching property nor the uniformity. It remains to translate (8) and the space bound into the two regimes.

For Item 1, $t \geq 2$ and the primes are fixed, so $\eta = \Theta_m(\ell^{t-1}) \to \infty$ and the case condition holds once ℓ exceeds a constant depending on the fixed primes; for smaller ℓ the family has constant size and the claims are trivial. We have $\log \eta = \Theta_m(\log \ell)$ and $\eta^{1/t} = \Theta_m(\ell^{1-1/t})$, so the right side of (8) is $\exp\big(O_m\big(\ell^{1/t}(\log \ell)^{1-1/t}\big)\big) = \exp\big(O_m\big(\ell^{1/t}(\log(\ell+2))^{1-1/t}\big)\big)$, as claimed. For the space bound: $\theta =$

$\Theta_m(\log \ell)$, so $\log(H+1) = O(\log w + \theta) = O(\log \ell)$ and $(mw)^{1/t} = \Theta_m\big(\ell/(\eta\theta)^{1/t}\big) = \Theta_m\big(\ell^{1/t}(\log \ell)^{-1/t}\big)$, whence

$$\begin{aligned}(mw)^{1/t}\log(H+1) &= O_m\big(\ell^{1/t}(\log \ell)^{1-1/t}\big) \\ &\le \ell && (\ell \text{ above a constant depending on the fixed primes}) \\ &\le \log N. && (N \ge 2^{\ell})\end{aligned}$$

Since $\delta \le (mw)^{1/t}$ (by the choice of exponents in the proof of Theorem 3.10) and $\log H \le \log(H+1)$, this also verifies $\log N \ge \delta \log H$, so the family is log-space uniform.

For Item 2, $t = \lceil \sqrt{\log \ell} \rceil$ and $p_1, \dots, p_t$ are the first t odd primes; the uniformity algorithm generates them by trial division in space $O(\log m)$, whereas in Item 1 the primes are fixed constants. By the prime number theorem $m = t^{t+o(t)}$, so $\log m = \Theta(\sqrt{\log \ell}\,\log\log \ell)$ and

$$\begin{aligned}\log \eta &= (t-1)\log \ell - \log m && (\text{definition of } \eta) \\ &= \Theta\big((\log \ell)^{3/2}\big), && (t = \lceil \sqrt{\log \ell} \rceil\text{; bound on } \log m)\end{aligned}$$

which is $\ge \Omega(t^2) = \Omega(\log \ell)$ for ℓ larger than an absolute constant. Then

$$\log\big(\eta^{1/t}\big) = \big(1 - \tfrac{1}{t}\big)\log \ell - \frac{\log m}{t} = \log \ell - \Theta\big(\sqrt{\log \ell}\big), \qquad \text{i.e.,} \qquad \eta^{1/t} = \ell \cdot 2^{-\Theta(\sqrt{\log \ell})},$$

so the right side of (8) is

$$\exp\left(O\left(\frac{\ell(\log \ell)^{3/2}}{\sqrt{\log \ell}\cdot \ell \cdot 2^{-\Theta(\sqrt{\log \ell})}}\right)\right) = \exp\Big(2^{O(\sqrt{\log \ell})}\Big) = \exp\big(\exp\big(O\big(\sqrt{\log \ell}\big)\big)\big),$$

as claimed. For the space bound: $\theta = \Theta(\log \eta/t^2) = \Theta(\sqrt{\log \ell})$, so $\log(H+1) = O(\log w + \theta) = O(\log \ell)$ and $(mw)^{1/t} = \Theta\big(\ell/(\eta\theta)^{1/t}\big) = 2^{O(\sqrt{\log \ell})}$, whence $(mw)^{1/t}\log(H+1) = 2^{O(\sqrt{\log \ell})} \le \ell \le \log N$ for ℓ larger than an absolute constant; as before this also verifies $\log N \ge \delta \log H$, so the family is log-space uniform. □