Electronic Colloquium on Computational Complexity, Report No. 22 (2026)

Catalytic Tree Evaluation From Matching Vectors

Alexandra Henzinger* Edward Pyne† Seyoon Ragavan‡

February 16, 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, a simple reduction from TreeEval to circuit evaluation, combined with the result of Buhrman et al. (STOC 2014), gives a catalytic algorithm for TreeEval running in logarithmic $O(\log n)$ free space and polynomial time, but with polynomial 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 $2^{\log^{\epsilon} n}$ catalytic space (for any $\epsilon > 0$). Our result opens a new line of attack on putting TreeEval in logspace, and immediately implies an improved simulation of time by *catalytic* space, by the reduction of Williams (STOC 2025).

Our catalytic TreeEval algorithm is inspired by a connection to matching-vector families and private information retrieval, and improved constructions of (uniform) matching-vector families would imply improvements to our algorithm.

*Email: ahenz@csail.mit.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 in part by Jane Street.

ISSN 1433-8092

Contents

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 to solve problems than is 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 is 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 to 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 information-theoretic cryptography. Building on this connection, we obtain a new catalytic algorithm for the tree evaluation problem [CMW+12], abbreviated $\mathsf{TreeEval}$. $\mathsf{TreeEval}$ has become a central protagonist in complexity theory for its roles in (1) aiming to separate polynomial time $\mathsf{P} = \mathsf{Time}(\mathsf{poly}(n))$ and logarithmic space $\mathsf{L} = \mathsf{Space}(O(\log n))$ [CMW+12, CM24] and (2) giving new tradeoffs between space and time [Wil25], namely that $\mathsf{Time}(T(n)) \subseteq \mathsf{Space}(\sqrt{T(n)\log T(n)})$. In Section 3, we show 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 3.2 for a parameterized statement.)*

By contrast, the threshold-circuit-evaluation algorithm of Buhrman et al. [BCK+14] implies a catalytic algorithm for $\mathsf{TreeEval}$ with superpolynomially more catalytic space: it uses $O(\log n)$ free space, $\mathsf{poly}(n)$ catalytic space, and polynomial runtime. At the same time, a brilliant 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 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 3.3 for a parameterized statement.)*

Implications. Our new catalytic algorithm for tree evaluation has two major direct implications:

1. *A new line of attack on Tree Evaluation.* Our technique can be thought of as a generalization of the techniques in the breakthrough result of Cook and Mertz, allowing us to apply technical tools developed in a cryptographic context. We view this as providing a direct line of attack on $\mathsf{TreeEval}$ (as we discuss later, improvements to these technical tools now directly imply better algorithms) and an indirect one (by building connections between tree evaluation, catalytic computation, and other areas).

2. *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. For every $\epsilon > 0$, a time $T = T(n)$ multitape Turing Machine can be decided in $O(\sqrt{T})$ free space, $2^{O(T^\epsilon)}$ catalytic space, and $2^{O(\sqrt{T})}$ time.

Relative to Williams' result, this shrinks the free space by a factor of $O(\sqrt{\log T})$ and shrinks the runtime by superpolynomial factors (from $2^{O(\sqrt{T \log T})}$ to $2^{O(\sqrt{T})}$ time), at the expense of introducing a large catalytic tape. Separately, combining the reduction of Williams with the catalytic circuit-evaluation algorithm of Buhrman et al. [BCK+14] 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 this result along with some additional tradeoffs (including for circuit evaluation) in Section 5.

The core machinery driving our new catalytic $\mathsf{TreeEval}$ algorithm is a family of *matching vectors* [Gro00]; informally, this is a collection of vectors whose inner products 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 schemes in the few-server regime [DG16, GKS25]. Our result is the first use of such techniques in catalytic computing. We view this connection as the main contribution of the paper, and we are optimistic for further applications and connections between cryptography, coding theory, and catalytic computing.

For one direct example, there is a large gap between known upper and lower bounds for matching vector families, and improvements to constructions of these 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 particular:

Theorem 1.4 (Informal)**.** *Suppose that (sufficiently uniform) matching-vector families with parameters not ruled out by known lower bounds exist. Then* $\mathsf{TreeEval}$ *can be solved with* $O(\log n \log\log n)$ *free space and polynomial time. (See Remark 3.4.)*

1.1 Overview

The tree evaluation problem. We follow the presentation of Goldreich [Gol25]. The tree evaluation problem, $\mathsf{TreeEval}_{h,\ell}$, is the following task [CMW+12]: given a binary tree of height h,[1] 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 $u \in \{0,1\}^{<h}$. We note that the input length to the $\mathsf{TreeEval}_{h,\ell}$ problem is thus $n = 2^h \cdot \ell \cdot 2^{2\ell}$.

Our tree evaluation algorithm draws inspiration from information-theoretic cryptography, which we unpack for the remainder of this overview. Before that, we note that our final construction is presented in Section 3 and that the presentation therein is completely self-contained.

Inspiration from cryptography. Our improved algorithm for $\mathsf{TreeEval}$ takes inspiration from cryptography. This new connection is based on the following view: both catalytic computing and cryptographic protocols work by operating on *masked* values. In particular, catalytic computation requires intermediate

[1] The tree evaluation problem was initially defined with tree fanin $r \geq 2$. For clarity, we focus on the $r = 2$ case; there is a simple reduction from any constant r to $r = 2$ (see Lemma 2.7) that suffices for the application of [Wil25].

results to be written onto the catalytic tape, meaning these results are stored and accessed while masked by the arbitrary contents of the catalytic tape τ. (Though it is tempting to overwrite the catalytic tape's content τ, this would not be a reversible transformation, likely breaking the contract of being able to restore the tape to τ at the end of the computation.) Similarly, cryptography shows how to privately outsource a computation to untrusted parties by masking the computation's inputs with randomness.

We make this connection more precise in the special case of $\mathsf{TreeEval}_{h,\ell}$ and information-theoretic private information retrieval (PIR). A PIR protocol [CGKS95, CGKS98] is defined with respect to a database $\mathsf{DB} \in \mathcal{R}^{n_{\mathsf{DB}}}$, which holds n_{DB} records that are each elements in some ring $\mathcal{R}$, and a number of servers $s \geq 2$. The protocol allows a user to read a record from the database DB, which is stored on the s servers, without revealing to any of the servers what record was read. Informally, a PIR protocol consists of three algorithms:

1. $\mathsf{Query}(i \in [n_{\mathsf{DB}}]) \to \mathsf{qu}_1, \ldots, \mathsf{qu}_s$, which the user runs on the index i that it wants to read, to produce s PIR queries, each of which is sent to one of the s PIR servers.

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

3. $\mathsf{Reconstruct}(i, \mathsf{ans}_1, \ldots, \mathsf{ans}_s) \to \mathcal{R}$, which the user runs on its index i and on the PIR answers from each server, to recover the i^{th} record of the database DB.

The *privacy* requirement in PIR guarantees that the marginal distribution of each qu_j, for $j \in [s]$, is independent of the index i queried. In many PIR schemes [CGKS95, BIM00, WY05], this is achieved by additively masking the index i (encoded as an element of some vector space) with uniformly-sampled randomness.

Connecting tree evaluation to private information retrieval. Consider a particular node u in the $\mathsf{TreeEval}$ tree. The algorithm's task at this node is to evaluate the function $f_u(v_{u0}, v_{u1})$ or, equivalently, to retrieve the entry corresponding to index $v_{u0} \| v_{u1}$ from the truth table of f_u (which consists of $2^{2\ell}$ records in $\{0,1\}^\ell$). The challenge is that the algorithm must do this *without* seeing either of v_{u0}, v_{u1} in the clear: to save space, the algorithm stores these values on the catalytic tape and accesses them masked by the tape's arbitrary initial contents.

This work starts by taking the view that the Cook-Mertz algorithm for $\mathsf{TreeEval}$ [CM24] can be seen as solving this task by making white-box use of an s-server PIR protocol. Very roughly, this use of PIR allows for retrieving any entry in the truth table of the function f_u by making s calls to the Answer algorithm—in a way that none of these s calls gets to see the index being retrieved. The $\mathsf{TreeEval}$ procedure runs each of these s calls to Answer (simulating each of the s PIR servers) in sequence and, in between, makes recursive calls on the child nodes to prepare the corresponding PIR queries (output by the Query algorithm) on the catalytic tape. Finally, using the $\mathsf{Reconstruct}$ algorithm, the $\mathsf{TreeEval}$ procedure can recover the value $f_u(v_{u0}, v_{u1})$ on the catalytic tape—as always, masked by the catalytic tape's initial contents.

This mapping from private information retrieval to catalytic computation of a function f_u is a conceptual link, intended as an intuitive (rather than a formal) correspondence. Making it precise requires imposing a number of structural requirements on the PIR scheme, which we describe in Section 4. An immediate distinction (see Remark 4.3) is that PIR usually masks queries uniformly at random, while in $\mathsf{TreeEval}$ 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. Furthermore, we require a PIR scheme that can be massaged to

1. run the Answer and $\mathsf{Reconstruct}$ methods interleaved with each other to save space,

2. have the PIR queries $\mathsf{qu}_1, \ldots, \mathsf{qu}_s$ each take the form $\tau + \mathsf{encode}_j(i)$ for some random value τ (which will be our catalytic tape) and some deterministic function encode_j applied to the index i being queried, for $j \in [s]$, and

3. allow the servers to "concatenate" PIR queries for values v_{u0} and v_{u1} into a PIR query for $v_{u0} \| v_{u1}$.

Then, this mapping from PIR to TreeEval requires running PIR over a database that is not exactly the truth table of f_u, but instead that maps each value in the truth table of f_u to a PIR query for that index (which lets us prepare the PIR query for the next layer of the tree). Nonetheless, we view this connection as a useful abstraction for describing our scheme, and as an open route to obtaining more advances in catalytic computing by building on cryptographic techniques. Towards this end, in Section 4, we formally define a new primitive, which we call *catalytic information retrieval*, that allows us to take a unified view of the algorithm of [CM24] and our work.

Which PIR scheme to use? The PIR protocol embedded in Cook-Mertz [CM24] relies on classic Reed-Muller codes [CGKS95, WY05]. However, we show that this is not the only option: in this work, we make white-box use of the most communication-efficient PIR protocols known to date, based on matching-vector codes [DG16, GKS25]. The advantage of these PIR schemes is that they can achieve subpolynomial communication with only a *constant* number of servers, whereas Reed-Muller PIR requires a superconstant number of servers to do so. (In particular, Cook-Mertz uses Reed-Muller PIR with $s = \mathsf{poly}(\log n_{\mathsf{DB}}) = \mathsf{poly}(\ell)$ servers.) The benefit of having fewer servers is that our TreeEval algorithm requires only a constant number of recursive calls at each level of the tree, and so only a constant amount of free space at each node to track this call stack. As a result, we obtain improvements in both the required amount of free space and runtime. However, a limitation is that matching-vector PIR with a constant number of servers has larger communication than Reed-Muller PIR with $\mathsf{poly}(\log n_{\mathsf{DB}})$ servers; correspondingly, our new TreeEval algorithm requires a larger catalytic tape, which Cook-Mertz does not.

To be more precise, when running TreeEval over a tree with height h and ℓ-bit labels using a "suitable" PIR protocol (as described above), that has s servers and communication complexity CC, we recover a TreeEval algorithm that requires:

- $O(s)$ recursive calls at each node,
- $O(h \log s)$ free space to track which recursive call is being executed at each node in the call stack,
- $O(\mathsf{CC})$ catalytic space to store PIR queries and answers as they are being computed, and
- additional $O(\log \mathsf{CC} + \ell)$ free space, to stream through the database while answering PIR queries (note that this cost is only incurred at one node at a time).

In sum, ignoring low-order terms, the free space of the TreeEval algorithm would be $O(h \log s + \log \mathsf{CC} + \ell)$, the catalytic space would be $O(\mathsf{CC})$, and the runtime would be $\mathsf{poly}(s^h \cdot 2^\ell)$. This view recovers the result of Cook and Mertz [CM24, Gol25] by simply regarding the catalytic space as true space, and using Reed-Muller PIR with $s = \mathsf{poly}(\ell)$ servers and $\mathsf{CC} = O(\log \ell)$ communication, giving a TreeEval algorithm that requires $O(h \log \ell + \ell)$ space and $\mathsf{poly}(\ell^h \cdot 2^{\ell+h})$ time.

Instead, as mentioned above, our new TreeEval algorithm uses low-communication PIR protocols based on matching-vector families. For any constant parameter $t \geq 1$, these schemes use 2^t servers, while

achieving communication complexity $\mathsf{CC} = \exp(\widetilde{O}((\log n_{\mathsf{DB}})^{1/t}))$ [Efr12, DG16, GKS25].[2] As a result, our true space goes down to $O(h + \ell)$ and our time to $\mathsf{poly}(2^{h+\ell})$. (The $\log \mathsf{CC}$ term in the true space is lower-order.) The tradeoff is that the catalytic space (equivalent to the PIR scheme's communication) is no longer logarithmic; instead, it grows to be subpolynomial in the input length: $\exp(\widetilde{O}(\ell^{1/t}))$. Making this algorithm work requires heavy white-box use of these PIR protocols. We defer these technical details to Section 3, and provide some additional discussion on the relation to PIR in Section 4. We describe applications to new tradeoffs between time, space, and catalytic space, obtained via the reduction of [Wil25], in Section 5.

1.2 Related Work

Algorithms for Tree Evaluation. There has been extensive work on algorithms [Bar89, BC92, CMW+12, CM20, 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 (apart from the algorithm implicit in the work of [BCK+14]), showing that subpolynomial catalytic space suffices to use only logarithmic free 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 we could evaluate any size-n circuit using $O(\log n)$ free space and $\mathsf{poly}(n)$ catalytic space.

Matching Vectors. Yekhanin [Yek08] first used matching vectors to obtain new families of locally decodable codes, starting a cascade of subsequent works [KY09, Efr09, IS10, DGY11, Yek12]. Since then, matching-vector codes have found applications in low-communication PIR [DG16, GKS25] 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].

1.3 Open Questions

Our work leaves open a number of questions. The most immediate is whether $\mathsf{TreeEval}$ is indeed in classical logspace, or in simultaneous polynomial time and $\log^c n$ space for $c < 2$. Building on this work, one approach to showing $\mathsf{TreeEval} \in \mathsf{L}$ would be to design a procedure for materializing matching vectors "on the fly" from a much more succinct representation stored in catalytic space. This could bring down the catalytic-space usage of our algorithm—perhaps all the way to $O(\log n)$ total space.

A second open question is whether known lower bounds on families of PIR schemes or of locally-decodable codes translate to lower bounds on $\mathsf{TreeEval}$. For example, a number of 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 $\mathsf{TreeEval}$—or, alternatively, to use techniques from the PIR literature for circumventing this bound to speed up $\mathsf{TreeEval}$.

[2] In fact, [DG16, GKS25] show even better tradeoffs between the communication and the number of servers. [DG16] achieve the stated communication with just 2^{t-1} servers and [GKS25] show that this can be improved for certain numbers of servers using S-decoding polynomials [CFL+13]. Though the results by [Efr12] suffice for our purposes, these techniques would yield fine-grained improvements to our theorems.

Finally, we leave open whether it is possible to port other techniques from cryptography to obtain further improvements in catalytic computing.

2 Preliminaries

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

2.1 The Catalytic Space Model

We first define catalytic machines:

Definition 2.1. 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 2.2. Let $\mathsf{CatTimeSpace}[C(n), S(n), T(n)]$ 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 2.3 (Representations of rings on the catalytic tape). Our algorithms (and those of Cook and Mertz and many other works) interpret the catalytic tape as holding a vector in $\mathbb{Z}_m^{O(d)}$, though the definition of catalytic space gives a catalytic tape that consists of bits. This translation requires some care: for example, if we naively represent $\mathbb{Z}_3$ with 2 bits, the catalytic tape could consist of values which do not correspond to an element of the ring. However, we can deal with this with a standard trick [CP25] that increases the space to represent each element in $\mathbb{Z}_m$ to $O(\log(dm))$ bits and increases the runtime by an additive $\mathsf{poly}(md)$. If our initial registers are $\tau_1, \dots, \tau_{O(d)} \in \mathbb{Z}_m$, we search for an offset $\Delta \in \{0,1\}^{O(\log dm)}$ such that $\tau_i + \Delta$ represents a valid entry in $\mathbb{Z}_m$ for every i. We store Δ during the computation, then subtract it from each register before halting.

2.2 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 2.4 ([CM24]). $\mathsf{TreeEval}_{\mathsf{h},\ell} \in \mathsf{Space}[O(\log(n) \cdot \log\log(n))]$.

A further result of [Sto23, Gol25] reduces the space by a $\log\log\log(n)$ factor.

Furthermore, 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 that:

Theorem 2.5 ([BCK+14]). $\mathsf{TreeEval}_{\mathsf{h},\ell} \in \mathsf{CatTimeSpace}\left[n^c, O(\log n), n^c\right]$.

Remark 2.6. This result follows from reducing $\mathsf{TreeEval}_{h,\ell}$ to a circuit and then evaluating such a circuit in CL. However, the smallest circuit class known to contain $\mathsf{TreeEval}_{h,\ell}$ (log-depth unbounded-fanin circuits) also contains a complete problem for non-deterministic logspace NL. Thus, improving the catalytic space of the latter step even to $n^{1-\varepsilon}$ is ruled out by conjectured time-space tradeoffs for NL [CP25].

Finally, we can reduce from TreeEval with constant (but greater than 2) fanin to tree eval with fanin 2, at a mild cost to the height.

Lemma 2.7. *There is a space $O(h \log r + \ell r)$-reduction from $\mathsf{TreeEval}_{h,\ell,r}$ to $\mathsf{TreeEval}_{h\lceil \log r \rceil, \ell \cdot \lceil r/2 \rceil, 2}$.*

Proof. We set $\ell' = \ell \cdot \lceil r/2 \rceil$. For every node u in the original tree with children $u_1, \dots, u_r$, we place a tree gadget with r leaves with height $\lceil \log r \rceil$. We think of the values at leaves $u_1, \dots, u_r$ as ℓ-bit strings padded by $0^{\ell \lceil r/2 \rceil - \ell}$. All non-root nodes in the gadget tree simply collect the values passed from their children, using that the output is of sufficient length to propagate both. The root node in the gadget, which has $v_{u_1}, \dots, v_{u_r}$ as input, has output $f_u(p_{u_1}, \dots, p_{u_r})$ padded by $0^{\ell \lceil r/2 \rceil - \ell}$. This transformation is clearly logspace uniform in the size of the resulting tree, and preserves the value at the root. □

2.3 Matching Vector Families

We next define matching-vector families and verify some useful facts about them. Let $p_1, p_2, \dots, p_t$ be t distinct odd primes. Let $m = p_1 \dots p_t$, and let $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ denote the ring of integers modulo m.

Definition 2.8. For $v_1 \in \mathbb{Z}_{p_1}, \dots, v_t \in \mathbb{Z}_{p_t}$, we let $\mathsf{CRT}(v_1, \dots, v_t)$ denote the unique value $v \in \mathbb{Z}_m$ such that $v \equiv v_i \pmod{p_i}$ for $i \in [t]$.

Definition 2.9. Let d be a positive integer. We say that two collections of vectors (U, V), where $U = (\mathbf{u}_1, \dots, \mathbf{u}_N) \in (\mathbb{Z}_m^d)^N$ and $V = (\mathbf{v}_1, \dots, \mathbf{v}_N) \in (\mathbb{Z}_m^d)^N$, form a matching-vector family over $\mathbb{Z}_m^d$ of size N, if:

1. for every $i, j \in [N]$, it holds that $\langle \mathbf{u}_i, \mathbf{v}_j \rangle \in \{0, 1\} \bmod p_k$ for $k \in [t]$, and
2. for every $i, j \in [N]$, we have $i = j$ if and only if $\langle \mathbf{u}_i, \mathbf{v}_j \rangle = 1$.

Definition 2.10. We say such a (sequence of) families $\{U, V\}_{N \in \mathbb{N}}$ is logspace uniform if there is an algorithm that, on input $(1^N, i, j, p_1, \dots, p_t)$ 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 2.11. For convenience, our definition of matching-vector families differs in a minor way from the customary definition [Efr12, DG16]: rather than having Item 2 say that the inner product of $\mathbf{u}_i$ with $\mathbf{v}_j$ is 0 iff $i = j$, we say that this inner product is 1. These definitions are equivalent, up to increasing the dimension d by one: if we define $\mathbf{u}'_i = (\mathbf{u}_i \| 1)$ and $\mathbf{v}'_j = (-\mathbf{v}_j \| 1)$, then it holds that $\langle \mathbf{u}'_i, \mathbf{v}'_j \rangle = 1 - \langle \mathbf{u}_i, \mathbf{v}_j \rangle$. As a result, Item 1 is preserved by this transformation, and moreover we have $\langle \mathbf{u}_i, \mathbf{v}_j \rangle = 0 \Leftrightarrow \langle \mathbf{u}'_i, \mathbf{v}'_j \rangle = 1$.

A crucial ingredient to our constructions is the following theorem:

Theorem 2.12 ([BBR94, Gro00, DGY11])**.** *Let m be the product of t distinct primes $p_1, \dots, p_t$. Let w be a positive integer. Then there exists a logspace-uniform matching-vector family over $\mathbb{Z}_m^d$ of size N, where:*

$$N := \binom{\lceil w^{1+1/t} \rceil}{w}, \text{ and}$$

[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.

$$d := 1 + \sum_{j=0}^{\lfloor (mw)^{1/t} \rfloor} \binom{\lceil w^{1+1/t} \rceil}{j}.$$

Proof Sketch. This is immediate from setting parameters suitably in [DGY11, Lemma 11]. In their theorem, we set $h = \lceil w^{1+1/t} \rceil$ and for each $i \in [t]$ take e_i to be minimal such that $p_i^{e_i} > (mw)^{1/t}/p_i$. This implies that $\prod_i p_i^{e_i} > mw/\prod_i p_i = w$, and moreover for any i we have $p_i^{e_i} \leq (mw)^{1/t}$. Finally, [DGY11, Lemma 11] does not state logspace-uniformity but it is immediate from their construction. We verify this in Section A. □

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

Corollary 2.13. All of the following hold:

1. ([Gro00]) Let t be constant and $p_1, \ldots, p_t$ be t fixed odd primes. Then for any $\ell \geq 1$, there exists a logspace-uniform matching-vector family over $\mathbb{Z}_{p_1 p_2 \cdots p_t}$ of size 2^ℓ with dimension $d = \exp\left(O\left(\ell^{1/t}(\log \ell)^{1-1/t}\right)\right)$.
2. For any $\ell \geq 1$, let $t = \sqrt{\log \ell - \log\log \ell/2 + O(1)}$, and let $p_1, \ldots, p_t$ be the first t odd primes. Then there exists a logspace-uniform matching-vector family over $\mathbb{Z}_{p_1 p_2 \cdots p_t}$ of size 2^ℓ with dimension $d = \exp(\exp(O(\sqrt{\log \ell})))$.

3 Main Result: Catalytic Tree Evaluation from Matching Vectors

We now prove the following theorem:

Theorem 3.1. *Let odd primes $p_1, \ldots, p_t$ be given as input and let $m = \prod_i p_i$. Suppose there exists an $O(\ell)$-space uniform matching vector family of size 2^ℓ over $\mathbb{Z}_m^d$, and suppose additionally that $d \log m \leq \mathsf{poly}(2^{h+\ell})$. Assume our algorithm is given the following resources:*

- *a catalytic tape of length $O(d \log(dm))$.*
- *on the input tape, 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 inputs $v_u \in \{0,1\}^\ell$ for $u \in \{0,1\}^h$.*

Then, there exists an algorithm that uses $O(\ell + h \log m)$ free space and time $\mathsf{poly}(2^{\ell+ht})$ and outputs $v_\emptyset$ (i.e., the result of TreeEval*).*

We can then combine this result with the matching-vector families given by Corollary 2.13 to read off the results in the introduction:

Corollary 3.2. 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. This follows by taking Item 1 of Corollary 2.13 with t a sufficiently large constant, e.g., $t = \lceil 3/\epsilon \rceil$. □

Corollary 3.3. 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. This follows from Item 2 of Corollary 2.13. □

Remark 3.4 (Better tree evaluation from better matching-vector families). There is a wide gap between the best-known constructions and lower bounds for matching-vector families. To the best of our knowledge, it would be consistent with current lower bounds [TB98, BDL14, ADL+25, GGMT25] for there to be matching-vector families (with $t, p_1, \ldots, p_t$ all constant) of size 2^ℓ and dimension $O(\ell \log \ell)$.[4] If such matching-vector families were to exist, then Theorem 3.1 would imply an algorithm for TreeEval that simultaneously uses $O(\log n \log\log n)$ space—matching [CM24]—and runs in polynomial time.

3.1 Recursive Step: One Level of Tree Evaluation

The main technical workhorse for our results is the following theorem:

Theorem 3.5. *Let $m = p_1 p_2 \ldots p_t$ be a product of t distinct odd primes that are given as input. Suppose there exists an $O(\ell)$-space uniform matching vector family $\{\mathbf{u}_x, \mathbf{v}_x \in \mathbb{Z}_m^d : x \in \{0,1\}^\ell\}$ of size 2^ℓ over $\mathbb{Z}_m^d$. Additionally, let $\{\mathbf{w}_s : s \in \{0,1\}^\ell\}$ be any $O(\ell)$-space uniform collection of vectors in $\mathbb{Z}_m^d$. Suppose our algorithm is given the following resources:*

- *global space comprising three registers $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{Z}_m^d$;*
- *the truth table of a function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$ on the input tape; and*
- *an oracle $\mathcal{O}$ that takes as input a scalar $\gamma \in \mathbb{Z}_m$ and bits $\mathsf{ctrl}, \sigma \in \{0,1\}$ in local space and updates the registers as follows. In the below, $a, b \in \{0,1\}^\ell$ are some bitstrings.*
  - *if $\sigma = 0$, let Δ denote $\mathbf{u}_a$ if $\mathsf{ctrl} = 0$ and $\mathbf{v}_a$ otherwise. The oracle will update $\mathbf{x} \leftarrow \mathbf{x} + \gamma\Delta$ and leave all other registers unchanged. (Here, by $\mathbf{x}$ we mean the first of the three catalytic registers, and $\mathbf{y}$ refers to the second catalytic register.)*
  - *if $\sigma = 1$, let Δ denote $\mathbf{u}_b$ if $\mathsf{ctrl} = 0$ and $\mathbf{v}_b$ otherwise. The oracle will update $\mathbf{y} \leftarrow \mathbf{y} + \gamma\Delta$ and leave all other registers unchanged.*

Then, there exists an algorithm that takes as input a scalar $\gamma \in \mathbb{Z}_m$ in local space and updates $\mathbf{z} \leftarrow \mathbf{z} + \gamma \mathbf{w}_{f(a,b)}$ (while leaving the $\mathbf{x}$ and $\mathbf{y}$ registers unchanged). Moreover, the algorithm uses $O(\ell + \log m + \log(d \cdot \log m))$ local space, and before making all oracle calls erases all but $O(\log m)$ bits of this space. The algorithm runs in time $\mathsf{poly}(2^{\ell+t} \cdot d \cdot \log m)$ and makes $2^{O(t)}$ queries to $\mathcal{O}$.

Corollary 3.6. We will use three corollaries of this theorem:

- letting $\mathbf{w}_s = \mathbf{u}_s$, we can update $\mathbf{z} \leftarrow \mathbf{z} + \gamma \mathbf{u}_{f(a,b)}$;
- letting $\mathbf{w}_s = \mathbf{v}_s$, we can update $\mathbf{z} \leftarrow \mathbf{z} + \gamma \mathbf{v}_{f(a,b)}$; and
- letting $\mathbf{w}_s = s$ (with appropriate zero padding), we can update $\mathbf{z} \leftarrow \mathbf{z} + \gamma s$.

Before we prove the theorem, we begin with some preliminary lemmas:

Lemma 3.7. *Let p be prime. For any $g_1, g_2 \in \mathbb{Z}_p$, consider the polynomial:*

$$f(X) = X^{g_1 g_2 \bmod p} - X^{(g_1+1) g_2 \bmod p} - X^{g_1 (g_2+1) \bmod p} + X^{(g_1+1)(g_2+1) \bmod p} \in \mathbb{Z}[X].$$

This polynomial is nonzero. Moreover, its nonzero coefficients are all in the set $\{-2, -1, 1, 2\}$.

[4] It was shown by [BDL14] that under the polynomial Freiman-Ruzsa conjecture [Ruz99] over $\mathbb{Z}_m^d$, the inequality $2^\ell \le \exp(O(d/\log d)) \Rightarrow d \ge \Omega(\ell \log \ell)$ must hold (assuming m is constant). This conjecture was recently proven by [GGMT25].

Proof. Suppose for the sake of contradiction that this polynomial is zero. Then, we would necessarily have $f'(1) = 0$. However, we have

$$f'(1) \equiv g_1 g_2 - (g_1 + 1)g_2 - g_1(g_2 + 1) + (g_1 + 1)(g_2 + 1) \equiv 1 \pmod{p},$$

which is a contradiction. It follows that the polynomial is nonzero. In addition, it is apparent that any coefficients of this polynomial must be integers in the interval $[-2, 2]$, so the conclusion follows. □

Lemma 3.8. *We can compute and store both $\langle \mathbf{x}, \mathbf{v}_a \rangle$ and $\langle \mathbf{y}, \mathbf{v}_b \rangle$ using $O(\ell)$ local space, at most $O(\log m)$ local space during oracle calls, and 4 calls to $\mathcal{O}$. At the end, none of the global space registers will be changed.*

Proof. This follows from the standard catalytic computing approach for computing inner products:

1. Compute $\mathsf{tmp}_1 = \langle \mathbf{x}, \mathbf{y} \rangle$ and write it into local space.
2. Swap the $\mathbf{x}, \mathbf{y}$ registers. The global state is now $(\mathbf{y}, \mathbf{x}, \mathbf{z})$.
3. Use $\mathcal{O}$ with $\sigma = 0, \mathsf{ctrl} = 1, \gamma = 1$. The global state is now $(\mathbf{y} + \mathbf{v}_a, \mathbf{x}, \mathbf{z})$.
4. Compute the inner product $\mathsf{tmp}_2 = \langle \mathbf{y} + \mathbf{v}_a, \mathbf{x} \rangle$ and write it into local space.
5. Use $\mathcal{O}$ with $\sigma = 0, \mathsf{ctrl} = 1, \gamma = -1$ to return the global state to $(\mathbf{y}, \mathbf{x}, \mathbf{z})$.
6. Use $\mathcal{O}$ with $\sigma = 1, \mathsf{ctrl} = 1, \gamma = 1$ to update the global state to $(\mathbf{y}, \mathbf{x} + \mathbf{v}_b, \mathbf{z})$.
7. Compute the inner product $\mathsf{tmp}_3 = \langle \mathbf{y}, \mathbf{x} + \mathbf{v}_b \rangle$ and write it into local space.
8. Use $\mathcal{O}$ with $\sigma = 1, \mathsf{ctrl} = 1, \gamma = -1$ to update the global state to $(\mathbf{y}, \mathbf{x}, \mathbf{z})$.
9. Swap the $\mathbf{x}, \mathbf{y}$ registers.

Note that $\mathsf{tmp}_2 - \mathsf{tmp}_1 = \langle \mathbf{x}, \mathbf{v}_a \rangle$ and $\mathsf{tmp}_3 - \mathsf{tmp}_1 = \langle \mathbf{y}, \mathbf{v}_b \rangle$, so we are done. □

3.2 Proof of Theorem 3.5

We next give the algorithm that underlies Theorem 3.5, together with its proof of correctness and efficiency analysis. Except where stated, all arithmetic is carried out modulo m.

Algorithm. We use γ^* to denote the value of γ that is given as input, along with the input function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$.

1. Use Lemma 3.8 to compute $g_1 = \langle \mathbf{x}, \mathbf{v}_a \rangle$ and $g_2 = \langle \mathbf{y}, \mathbf{v}_b \rangle$.
2. For each $i \in [t]$, compute the polynomial $f_i(X) = X^{g_1 g_2 \bmod p_i} - X^{(g_1+1)g_2 \bmod p_i} - X^{g_1(g_2+1) \bmod p_i} + X^{(g_1+1)(g_2+1) \bmod p_i} \in \mathbb{Z}[X]$. By Lemma 3.7, this polynomial is nonzero over $\mathbb{Z}_{p_i}[X]$ and therefore over $\mathbb{Z}[X]$ as well. Let $\alpha_i X^{\beta_i}$ be the lexicographically first nonzero monomial in $f_i(X)$. Since all exponents in β_i are reduced mod p_i, we can regard β_i as an element of $\mathbb{Z}_{p_i}$. We know from the lemma that $\alpha_i \in \{-2, -1, 1, 2\}$ and hence it is coprime to m.
3. Repeat for all bits $b_1, \ldots, b_t, c_1, \ldots, c_t \in \{0,1\}$:

(a) Using two queries to $\mathcal{O}$, make the updates:

$$\mathbf{x} \leftarrow \mathbf{x} + \mathsf{CRT}(b_1, \dots, b_t) \cdot \mathbf{u}_a$$
$$\mathbf{y} \leftarrow \mathbf{y} + \mathsf{CRT}(c_1, \dots, c_t) \cdot \mathbf{u}_b.$$

(For example, the first update would be with $\sigma = 0, \mathsf{ctrl} = 0, \gamma = \mathsf{CRT}(b_1, \dots, b_t)$. Recall that $\mathsf{CRT}(b_1, \dots, b_t), \mathsf{CRT}(c_1, \dots, c_t)$ are scalars in $\mathbb{Z}_m$ and $\mathbf{u}_a, \mathbf{u}_b$ are vectors in $\mathbb{Z}_m^d$.)

(b) Repeat for all $r, s \in \{0,1\}^\ell$:

i. Compute $\langle \mathbf{x}, \mathbf{v}_r \rangle \cdot \langle \mathbf{y}, \mathbf{v}_s \rangle \bmod m$.

ii. If the result is equal to $\mathsf{CRT}(\beta_1, \dots, \beta_t)$, update

$$\mathbf{z} \leftarrow \mathbf{z} + \gamma^* \cdot \mathbf{w}_{f(r,s)} \cdot (-1)^{\sum_{i \in [t]} (b_i + c_i)} \cdot \left(\prod_{i \in [t]} \alpha_i \right)^{-1}.$$

(Here, all multiplicative inverses are computed mod m.)

(c) Using two queries to $\mathcal{O}$, restore:

$$\mathbf{x} \leftarrow \mathbf{x} + \mathsf{CRT}(-b_1, \dots, -b_t) \cdot \mathbf{u}_a$$
$$\mathbf{y} \leftarrow \mathbf{y} + \mathsf{CRT}(-c_1, \dots, -c_t) \cdot \mathbf{u}_b.$$

Proof of correctness. We prove correctness in a few steps:

Lemma 3.9. *For any $i \in [t]$ and $r, s \in \{0,1\}^\ell$, we have:*

$$\sum_{\substack{b_i, c_i \in \{0,1\} \\ \langle \mathbf{x} + b_i \cdot \mathbf{u}_a, \mathbf{v}_r \rangle \cdot \langle \mathbf{y} + c_i \cdot \mathbf{u}_b, \mathbf{v}_s \rangle \equiv \beta_i \pmod{p_i}}} (-1)^{b_i + c_i} = \begin{cases} \alpha_i, \ \textit{if } (r,s) = (a,b), \\ 0, \ \textit{if } \langle \mathbf{u}_a, \mathbf{v}_r \rangle \cdot \langle \mathbf{u}_b, \mathbf{v}_s \rangle \equiv 0 \pmod{p_i}, \\ \textit{arbitrary, otherwise.} \end{cases}$$

Proof. First we address the second case where at least one of the inner products is 0 mod p_i. Assume without loss of generality that $\langle \mathbf{u}_a, \mathbf{v}_r \rangle \equiv 0 \pmod{p_i}$; the other case is analogous. In this case, whether or not $\langle \mathbf{x} + b_i \cdot \mathbf{u}_a, \mathbf{v}_r \rangle \cdot \langle \mathbf{y} + c_i \cdot \mathbf{u}_b, \mathbf{v}_s \rangle \equiv \beta_i \pmod{p_i}$ is independent of the choice of b_i. Thus c_i ranges over $\delta \in \{0,1\}$ such that $\langle \mathbf{x}, \mathbf{v}_r \rangle \cdot \langle \mathbf{y} + \delta \cdot \mathbf{u}_b, \mathbf{v}_s \rangle \equiv \beta_i \pmod{p_i}$, and for each such δ there are two terms corresponding to $b_i = 0$ and $b_i = 1$. Pairing up terms corresponding to $(b_i, c_i) = (0, \delta)$ and $(1, \delta)$ (for each included δ) implies the conclusion.

In the first case, the condition $\langle \mathbf{x} + b_i \cdot \mathbf{u}_a, \mathbf{v}_r \rangle \cdot \langle \mathbf{y} + c_i \cdot \mathbf{u}_b, \mathbf{v}_s \rangle \equiv \beta_i \pmod{p_i}$ that we are summing over simplifies to $(g_1 + b_i)(g_2 + c_i) \equiv \beta_i \pmod{p_i}$, noting that by the matching vector guarantee we have $\langle \mathbf{u}_a, \mathbf{v}_r \rangle \equiv \langle \mathbf{u}_b, \mathbf{v}_s \rangle \equiv 1 \pmod{p_i}$. Now, we note that the polynomial $f_i(X)$ can also be written as

$$\sum_{b_i, c_i \in \{0,1\}} (-1)^{b_i + c_i} \cdot X^{(g_1 + b_i)(g_2 + c_i) \bmod p_i}.$$

Thus, our expression of interest is the coefficient of X^{β_i} in $f_i(X)$, which is α_i by construction. The conclusion follows. □

Lemma 3.10. *For any $r, s \in \{0,1\}^\ell$, we have:*

$$\sum_{\substack{b_1, \dots, b_t, c_1, \dots, c_t \in \{0,1\} \\ \langle \mathbf{x} + b_i \cdot \mathbf{u}_a, \mathbf{v}_r \rangle \cdot \langle \mathbf{y} + c_i \cdot \mathbf{u}_b, \mathbf{v}_s \rangle \equiv \beta_i \pmod{p_i} \forall i \in [t]}} (-1)^{\sum_{i \in [t]} (b_i + c_i)} = \begin{cases} \prod_{i \in [t]} \alpha_i, \ \textit{if } (r,s) = (a,b), \\ 0, \ \textit{else.} \end{cases}$$

Proof. We can start by factoring over the independent choices of b_i, c_i for each prime p_i to obtain:

$$\sum_{\substack{b_1,\ldots,b_t,c_1,\ldots,c_t\in\{0,1\} \\ \langle \mathbf{x}+b_i\cdot\mathbf{u}_a,\mathbf{v}_r\rangle\cdot\langle \mathbf{y}+c_i\cdot\mathbf{u}_b,\mathbf{v}_s\rangle\equiv\beta_i \pmod{p_i}\forall i\in[t]}} (-1)^{\sum_{i\in[t]}(b_i+c_i)} = \prod_{i\in[t]}\left[\sum_{\substack{b_i,c_i\in\{0,1\} \\ \langle \mathbf{x}+b_i\cdot\mathbf{u}_a,\mathbf{v}_r\rangle\cdot\langle \mathbf{y}+c_i\cdot\mathbf{u}_b,\mathbf{v}_s\rangle\equiv\beta_i \pmod{p_i}}} (-1)^{b_i+c_i}\right].$$

If $r \neq a$, then by the matching vector guarantee there must exist an i such that $\langle \mathbf{u}_a, \mathbf{v}_r\rangle \equiv 0 \pmod{p_i}$. The corresponding term in the above product will be 0 by Lemma 3.9, which will make the entire product 0. We may argue similarly if $s \neq b$.

If $(r, s) = (a, b)$, then by Lemma 3.9, the i^{th} term in the above product is α_i, implying the conclusion. □

Correctness of our algorithm is then immediate from the following corollary:

Corollary 3.11. We have:

$$\left(\prod_{i\in[t]}\alpha_i\right)\cdot\mathbf{w}_{f(a,b)} = \sum_{r,s\in\{0,1\}^\ell}\mathbf{w}_{f(r,s)}\cdot\left[\sum_{\substack{b_1,\ldots,b_t,c_1,\ldots,c_t\in\{0,1\} \\ \langle \mathbf{x}+b_i\cdot\mathbf{u}_a,\mathbf{v}_r\rangle\cdot\langle \mathbf{y}+c_i\cdot\mathbf{u}_b,\mathbf{v}_s\rangle\equiv\beta_i \pmod{p_i}\forall i\in[t]}} (-1)^{\sum_{i\in[t]}(b_i+c_i)}\right].$$

Proof. By Lemma 3.10, the left-hand side and right-hand side are identical linear forms in the collection $\{\mathbf{w}_{r,s} : r, s \in \{0,1\}^\ell\}$. □

Efficiency analysis. The stated runtime guarantee is clear. (The $\mathsf{poly}(d \cdot \log m)$ factor is to allow for basic arithmetic operations in the $\mathbf{x}, \mathbf{y}, \mathbf{z}$ registers.)[5]

It remains to tally up the space needed at each point in the computation:

- In step 1, we compute and store g_1, g_2 in clean local space (of which we need $O(\log m)$).
- In step 2, we compute and store the coefficients α_i, β_i in local space $O(t + \sum_{i\in[t]}\log p_i) = O(\log m)$, which we persist across the entire computation.
- In step 3, we use $O(t) = O(\log m)$ local space to store the bits $b_1, \ldots, b_t$ and $c_1, \ldots, c_t$. Then, in steps (a) and (c) in this loop, this is all of the information we store. During step (b), where we use $O(\ell + \log(d \cdot \log m))$ local space to keep track of the values r and s (each of bitlength ℓ) and to keep pointers into registers $\mathbf{x}, \mathbf{y}, \mathbf{z}$, we do not invoke the oracle. □

Remark 3.12 (An alternate view in terms of polynomial rings). The reader might rightly suspect that our algorithm arose from a more complicated construction over the polynomial ring $\mathcal{R} := \mathbb{Z}_m[X_1, \ldots, X_t]/(X_1^{p_1} - 1, X_2^{p_2} - 1, \ldots, X_t^{p_t} - 1)$, akin to those of [DGY11, DG16, GKS25]. In this setting, our registers would be in $\mathcal{R}^d$ rather than $\mathbb{Z}_m^d$, and the update to the $\mathbf{z}$ register can be thought of as adding some multiple of

$$\prod_{i=1}^{t}\left(\sum_{b_i,c_i\in\{0,1\}}(-1)^{b_i+c_i}X_i^{(g_1+b_i)(g_2+c_i)}\right) = \prod_{i=1}^{t}f_i(X_i).$$

The resulting algorithm keeps track of unnecessary information (and makes unnecessary changes to the catalytic tape); we only ever need one monomial of a polynomial in $\mathcal{R}$, and the algorithm we present arises from making this simplification.

[5] Recall that arithmetic mod m can be computed in polynomial time and linear space in the input representation, i.e., $O(\log m)$.

3.3 Proof of Theorem 3.1

Finally, we use Theorem 3.5 in the natural recursive fashion to prove Theorem 3.1. For $u \in \{0,1\}^{\le h}$ specifying a node in the tree, recall that v_u is the value of the TreeEval instance at that node.

We instantiate a single free space register $u \in \{0,1\}^{\le h}$ to track the current location in the tree, which we initialize to $\emptyset$ (corresponding to the root node). We allocate $O(\ell + t + \log m + \log(d \log m)) = O(\ell + h + \log m)$ bits of free workspace to be used temporarily by the algorithm of Theorem 3.5 between its oracle calls, and $h \cdot O(\log m)$ free space allocated for storing the $O(\log m)$ free space used by the algorithm at each level while making oracle calls.

We interpret the catalytic tape as holding registers $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{Z}_m^d$. To handle that the catalytic tape consists of bits and not elements of $\mathbb{Z}_m$, we use Remark 2.3 (and the catalytic tape thus has length $O(d \log(dm))$). We store the final $\lceil \ell / \log m \rceil$ coordinates of $\mathbf{z}$ using free space, which we denote reg.

Finally, we invoke Theorem 3.5 at the root node with $\gamma^* = 1$ and $\mathbf{w}_s = s$ (where we cast $s \in \mathbb{Z}_m^{\lceil \ell / \log m \rceil}$ and pad to the appropriate length). We discuss how to handle oracle calls made by the one-level algorithm below. Once this procedure halts, we have that $\mathbf{z}$ is in configuration $\mathsf{reg} + v_\emptyset$ (and $\mathbf{x}, \mathbf{y}$ are unmodified), so we subtract reg and obtain $v_\emptyset$ as desired. Lastly, we run the algorithm again with $\gamma^* = -1$ and $\mathbf{w}_s = s$. It is easy to see that after this the catalytic tape is entirely restored, so we halt and return $v_\emptyset$.

Handling oracle calls. Suppose the one-level algorithm corresponding to node u at level $i \in \{2, \dots, h\}$ makes a call to $\mathcal{O}$ with input γ, ctrl and bit σ. First, if $i = 2$ (so the call corresponds to a leaf node at layer 1 of the tree) we define $a = v_{u0}$ and $b = v_{u1}$. We directly use $O(\ell + \log(d \log m)) = O(\ell + h)$ local space to make the update to $\mathbf{x}$ or $\mathbf{y}$ specified by the oracle API.

Otherwise, store $\gamma, \mathsf{ctrl}, \sigma$ and the currently used free space of the algorithm in the $O(\log m)$ bits of free space allocated for level i. If $\sigma = 0$ we swap $\mathbf{x}, \mathbf{z}$, and if $\sigma = 1$ we swap $\mathbf{y}, \mathbf{z}$. Both swaps can be performed using $O(\log(d \log m)) = O(\ell + h)$ temporary free space (which we then erase). We update the global indicator of our current node to $u \leftarrow u\sigma$, and invoke the algorithm of Theorem 3.5 with

$$f = f_u \qquad \gamma^* = \gamma \qquad \mathbf{w} = \begin{cases} \mathbf{u} & \mathsf{ctrl} = 0 \\ \mathbf{v} & \mathsf{ctrl} = 1 \end{cases}$$

and note that the child algorithm can store which part of the matching-vector family it should apply with a single bit of free space. After the child algorithm returns, we again swap $\mathbf{x}, \mathbf{z}$ if $\sigma = 0$ and $\mathbf{y}, \mathbf{z}$ if $\sigma = 1$, and update u to reflect the current node. By the correctness of the child algorithm, we have that this procedure halts with the register update specified by the oracle API.

Correctness. By Theorem 3.5 and the fact that we implement the specified oracle API, when the algorithm is run at node u with γ^* and vector family $\mathbf{w}$, it updates $\mathbf{z}$ to $\mathbf{z} + \gamma^* \cdot \mathbf{w}_{f(v_{u0}, v_{u1})} = \mathbf{z} + \gamma^* \cdot \mathbf{w}_{v_u}$. This establishes correctness by a simple inductive argument.

Runtime. Each single-level algorithm makes $2^{O(t)}$ oracle calls and runs in time $\mathsf{poly}(2^{\ell+t} \cdot d \log m) = \mathsf{poly}(2^{h+\ell+t})$, so an induction gives a final runtime $2^{O(th)} \cdot \mathsf{poly}(2^{h+\ell+t}) = \mathsf{poly}(2^{\ell+ht})$. □

4 An Alternate View: Catalytic Tree Evaluation from Private Information Retrieval

Next, we give an alternate presentation of Section 3's algorithm, phrased closer to the language of private information retrieval (PIR).

4.1 Background: Informal Definition of PIR

A PIR protocol is defined with respect to a database size $n_{\mathsf{DB}} \in \mathbb{N}$, a ring $\mathcal{R}$, and a number of servers $s \geq 2$. It consists of three polynomial-time algorithms:

1. $\mathsf{Query}(i) \to \mathsf{qu}_1, \ldots, \mathsf{qu}_s$, which takes as input an index $i \in [n_{\mathsf{DB}}]$ into a database and produces s PIR queries to be sent to each of the s servers.
2. $\mathsf{Answer}(\mathsf{DB}, \mathsf{qu}) \to \mathsf{ans}$, which takes as input a database $\mathsf{DB} \in \mathcal{R}^{n_{\mathsf{DB}}}$ and a PIR query qu and produces a PIR answer ans.
3. $\mathsf{Reconstruct}(i, \mathsf{ans}_1, \ldots, \mathsf{ans}_s) \to \mathcal{R}$, which takes as input the index being read and the s servers' answers and outputs the i^{th} record of the database DB.

The scheme's *privacy* requires the marginal distribution of each query qu_j, for $j \in [s]$, to be independent of the index i being queried.

4.2 Modifying the PIR Requirements for Tree Evaluation

We next show that, if a PIR scheme can be massaged into a tuple of algorithms with certain structural properties, then it can be used to build a catalytic algorithm for $\mathsf{TreeEval}$. At a high level, these properties correspond to the following intuitive requirements:

- The query routine samples some common randomness with which it additively masks a fixed sequence of elements (that depend only on the index being queried), one of which is sent to each server.
- The user can effectively make a query for a pair of indices $a \| b$ (in a larger database of size n_{DB}^2) by building a PIR query for a and a PIR query for b independently.
- The reconstruction functionality can be pulled into the Answer routine, given some small state that depends on the indices queried and on the randomness used. After this, reconstructing the record from each server's answer is just addition.
- All algorithms are low-space, and the servers can answer PIR queries by streaming over the database.

Definition of catalytic information retrieval. To be more formal, we define the syntax for a new object, which we call *catalytic information retrieval (CIR)*, to be the following tuple of three algorithms:

1. $\mathsf{DetQuery}(a \in [n_{\mathsf{DB}}], j \in [s], \mu \in \{0,1\}) \to \mathcal{R}$, a deterministic algorithm that takes in an index a into the database, a server $j \in [s]$, and a bit μ and produces the *deterministic* part of the query for the j^{th} server.

   In our scheme, the user makes queries to a tuple of indices $a \| b$ simultaneously. To do so, our user:

   - samples two ring elements $x, y \xleftarrow{R} \mathcal{R}$.
   - sends to server $j \in [s]$ the pair of PIR queries $\mathsf{qu}_j \leftarrow x + \mathsf{DetQuery}(a, j, 0)$ and $\mathsf{qu}'_j \leftarrow y + \mathsf{DetQuery}(b, j, 1)$.

2. $\mathsf{GetState}^{\mathcal{O}_{a,b}}(x \in \mathcal{R}, y \in \mathcal{R}) \to \mathsf{st} \in \{0,1\}^*$, a deterministic oracle algorithm that takes as input two registers holding the randomness x and y, and produces the state st needed for reconstruction.

   The oracle $\mathcal{O}_{a,b}$ takes as input a register $t \in \mathcal{R}$, bits $\sigma, \mu \in \{0,1\}$, a factor $\gamma \in \{-1, 1\}$, and a server index $j \in [s]$ and updates

$$t \leftarrow t + \gamma \cdot \mathsf{DetQuery}(c, j, \mu),$$

   where c is a if $\sigma = 0$, else it is b.

3. $\mathsf{AnswerAndReconstruct}(\mathsf{DB} \in \mathcal{R}^{n_{\mathsf{DB}}}, \mathsf{st}, j \in [s], \mathsf{qu} \in \mathcal{R}, \mathsf{qu}' \in \mathcal{R}) \to \mathsf{ans} \in \mathcal{R}$, a deterministic algorithm that takes as input the database DB, the reconstruction state st, the server index $j \in [s]$, and the two queries qu and qu', and outputs an answer ans.

We require a CIR scheme to satisfy two properties: correctness and efficiency.

Definition 4.1 (Correctness)**.** We require that for any $a, b \in [n_{\mathsf{DB}}]$ and $x, y \in \mathcal{R}$, it holds that:

$$\mathsf{DB}_{a\|b} = \sum_{j \in [s]} \mathsf{AnswerAndReconstruct}(\mathsf{DB}, \mathsf{GetState}^{\mathcal{O}_{a,b}}(x, y), j, x + \mathsf{DetQuery}(a, j, 0), y + \mathsf{DetQuery}(b, j, 1)).$$

Definition 4.2. We say a CIR scheme is *space-efficient* if each of the following are true:

- We can represent a valid element of $\mathcal{R}$ on the catalytic tape in space $\widetilde{O}(\log |\mathcal{R}|)$, and we can perform arithmetic operations in $\mathcal{R}$ in time $\mathsf{poly}\log |\mathcal{R}|$ and additional space $O(\log\log |\mathcal{R}|)$.
- $\mathsf{GetState}^{\mathcal{O}_{a,b}}$ is computable with catalytic registers x, y; it uses $O(\log n_{\mathsf{DB}})$ free space, and $O(|\mathsf{st}|)$ free space during every call to $\mathcal{O}$; and
- $\mathsf{AnswerAndReconstruct}$ is computable with $O(\log n_{\mathsf{DB}})$ free space and runs in time $\mathsf{poly}(n_{\mathsf{DB}})$, provided that we can random access into DB in $O(\log n_{\mathsf{DB}})$ space.

Remark 4.3. Jumping ahead to the setting of $\mathsf{TreeEval}$, note that in that setting the ring elements x, y will not be sampled at random; rather, they will be the contents of a possibly adversarially chosen catalytic tape.

This is why Definition 4.1 insists on perfect correctness. Our reason for describing a CIR scheme as sampling x, y is to be consistent with the typical PIR framework where each query needs to be marginally uniformly random.

4.3 Tree Evaluation Algorithm

We begin with the one-level algorithm for $\mathsf{TreeEval}_{h,\ell}$, following Section 3.1, recast in the language of catalytic information retrieval.

Theorem 4.4. *Assume we have a space-efficient CIR scheme. Suppose our algorithm is given the following resources:*

- *global space comprising registers $x, y, z \in \mathcal{R}$; and*
- *the truth table of a function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$ on the input tape;*
- *the oracle $\mathcal{O}_{a,b}$ (as defined in Item 2 of Section 4.2).*

Then, there exists an algorithm that takes as input a scalar $\gamma^* \in \{-1, +1\}$, a bit $\mu^* \in \{0, 1\}$, and an index $j^* \in [s]$ in local space and updates

$$z \leftarrow z + \gamma^* \cdot \mathsf{DetQuery}(f(a, b), j^*, \mu^*),$$

while leaving the x and y registers unchanged. Moreover, the algorithm uses $O(\ell + \log|\mathcal{R}| + |\mathsf{st}| + \log s)$ local space and before making all oracle calls erases all but $O(|\mathsf{st}| + \log s)$ bits of this space, makes $O(s)$ queries to $\mathcal{O}_{a,b}$, and runs in time $\mathsf{poly}(2^\ell, \log|\mathcal{R}|, s, 2^{|\mathsf{st}|})$.

Proof. We sketch the algorithm below and omit proofs of efficiency since they closely follow the proof of Theorem 3.5.

1. Let DB be the $2^{2\ell}$-record database that, in position $(r\|s)$, contains the record computed as

$$\gamma^* \cdot \mathsf{DetQuery}(f(r, s), j^*, \mu^*).$$

2. Compute $\mathsf{st} \leftarrow \mathsf{GetState}^{\mathcal{O}_{a,b}}(x, y)$, making oracle queries to $\mathcal{O}_{a,b}$. Then, store st in free space.

3. Repeat for each server indexed by $j \in [s]$:

   (a) Using two queries to $\mathcal{O}_{a,b}$, make the updates:

$$x \leftarrow x + \mathsf{DetQuery}(a, j, 0)$$
$$y \leftarrow y + \mathsf{DetQuery}(b, j, 1).$$

   (b) Using CIR with respect to the database defined in Item 1, update

$$z \leftarrow z + \mathsf{AnswerAndReconstruct}(\mathsf{DB}, \mathsf{st}, j, x, y)$$

   (c) Using two queries to $\mathcal{O}_{a,b}$, restore:

$$x \leftarrow x - \mathsf{DetQuery}(a, j, 0)$$
$$y \leftarrow y - \mathsf{DetQuery}(b, j, 1).$$

□

The next theorem readily follows by using the same recursive strategy as in Theorem 3.1.

Theorem 4.5. *Suppose the CIR scheme is space-efficient. Then, using the one-level algorithm above (where we implement the oracle $\mathcal{O}_{a,b}$ using a recursive instantiation of the algorithm in the natural way), we get an algorithm for $\mathsf{TreeEval}_{h,\ell}$ that uses $\widetilde{O}(\log|\mathcal{R}|)$ catalytic space, $O(h \cdot |\mathsf{st}| + h \log s + \ell)$ free space, and $O(s)^h \cdot \mathsf{poly}(2^\ell, \log|\mathcal{R}|, |\mathsf{st}|)$ runtime.*

Remark 4.6. We note that an even more general variant of a CIR scheme would still give new algorithms for $\mathsf{TreeEval}$. For example, $\mathsf{GetState}$ could also use the z register as catalytic space. Additionally, we do not need to work over a ring $\mathcal{R}$; we could work over an arbitrary universe and replace additions and subtractions with reversible updates. We refrain from formally presenting these abstractions for the sake of clarity.

4.4 Special Cases

The algorithm of Cook and Mertz. We now sketch how the algorithm of Cook and Mertz [CM24] can also be viewed as arising from a CIR scheme. For clarity, we will assume $n_{\mathsf{DB}} = 2^{2\ell}$ and that DB is viewed as the truth table of a function $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell$. We make the following choices:

- $\mathbb{F}$ will be a prime field of order $O(\ell)$ that has a primitive sth root of unity ω for some $s \in (2\ell, |\mathbb{F}|)$;
- The ring $\mathcal{R}$ will be $\mathbb{F}^\ell$;
- The number of servers will be s; and
- $g : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ will be the multilinear extension of f.

We now define the CIR scheme as follows:

- $\mathsf{DetQuery}(a, j, \mu) := \omega^{-j} a$ (note that μ is irrelevant for this construction);
- $\mathsf{GetState}$ will not do anything, i.e., $\mathsf{st} = \perp$;
- $\mathsf{AnswerAndReconstruct}(\mathsf{DB}, \mathsf{st}, j, x, y) := g(\omega^j x, \omega^j y)/m$. Here, $g(\cdot)$ is computed on the fly in space $O(\ell)$.

Since the number of servers is $O(\ell)$, this recovers the $O(\ell + h \log \ell)$ free space and $\widetilde{O}(\ell)$ catalytic space of [CM24, Theorem 1.3].[6]

The algorithm of Section 3. Here, the correspondence is easier to see because the presentation of our algorithm in this section is modeled off of Section 3. We sketch the correspondence below:

- The ring $\mathcal{R}$ is $\mathbb{Z}_m^{2d}$, hence we denote the catalytic registers $\mathbf{x}, \mathbf{y}, \mathbf{z}$ with boldface.
- There are 2^{2t} servers which we identify with strings of $2t$ bits $b_1, \dots, b_t, c_1, \dots, c_t$;
- $\mathsf{DetQuery}(a, j = (b_1, \dots, b_t, c_1, \dots, c_t), \mu) := \begin{cases} \mathsf{CRT}(b_1, \dots, b_t) \cdot (\mathbf{u}_a \| \mathbf{v}_a), \text{ if } \mu = 0 \\ \mathsf{CRT}(c_1, \dots, c_t) \cdot (\mathbf{u}_a \| \mathbf{v}_a), \text{ if } \mu = 1. \end{cases}$
- $\mathsf{GetState}$ will compute and store g_1, g_2, and all the α_i's and β_i's in st. In a little more detail, we will isolate $\mathbf{x}_{\mathsf{trunc}}, \mathbf{y}_{\mathsf{trunc}} \in \mathbb{Z}_m^d$ to be the first d entries of $\mathbf{x}, \mathbf{y}$ respectively and compute $g_1 = \langle \mathbf{x}_{\mathsf{trunc}}, \mathbf{v}_a \rangle$ and $g_2 = \langle \mathbf{y}_{\mathsf{trunc}}, \mathbf{v}_b \rangle$, which could potentially require some swaps within the $\mathbf{x}, \mathbf{y}$ registers that can be reversed.

  The oracle queries in Lemma 3.8 can be instantiated by setting $b_1 = \dots = b_t = c_1 = \dots = c_t = 1$, so that the factor coming from each CRT term is just ± 1.

- $\mathsf{AnswerAndReconstruct}(\mathsf{DB}, \mathsf{st}, j = (b_1, \dots, b_t, c_1, \dots, c_t), \mathbf{x}, \mathbf{y})$: this will be equal to

$$\sum_{r,s \in \{0,1\}^\ell} \begin{cases} \mathsf{DB}_{r\|s} \cdot (-1)^{\sum_{i \in [t]} (b_i + c_i)} \cdot \left(\prod_{i \in [t]} \alpha_i \right)^{-1}, \text{ if } \langle \mathbf{x}_{\mathsf{trunc}}, \mathbf{v}_r \rangle \cdot \langle \mathbf{y}_{\mathsf{trunc}}, \mathbf{v}_s \rangle \equiv \mathsf{CRT}(\beta_1, \dots, \beta_t) \pmod m, \\ 0, \text{ otherwise.} \end{cases}$$

[6] The algorithm of Cook-Mertz does not work in the catalytic space model, so they do not need to incur the space overhead of the transformation in Remark 2.3 (since they can initialize all registers to valid representations).

Here the number of servers is 2^{2t}, $|\mathsf{st}| = O(\log m)$, and $\log|\mathcal{R}| = O(d \log m)$. Plugging these in to Theorem 4.5 recovers the statement of Theorem 3.1, noting that we assume $d \log m \leq \mathsf{poly}(2^{h+\ell})$ and that the transformation of Remark 2.3 will only require catalytic space $O(d \log(dm))$ to represent an element of $\mathcal{R}$.

Motivating the algorithm of Section 3. We take the opportunity here to provide some high-level intuition for the various departures our algorithm in Section 3 makes from typical PIR protocols based on matching vector families [DGY11, DG16, GKS25, LBA25]. We start with the following simpler construction of 2^t-server PIR [Efr12] given a matching vector family of size N over $\mathbb{Z}_m^d$ (where $m = p_1 \dots p_t$ is a product of t primes). Let q be a prime such that $m|q-1$. Let $g_1, \dots, g_t \in \mathbb{Z}_q$ be elements with respective order $p_1 \dots p_t$. Servers are indexed by tuples $(b_1, \dots, b_t) \in \{0,1\}^t$ of bits. The protocol now proceeds as follows:

- Suppose the client has an index $i^* \in [N]$. They will sample uniformly random $\mathbf{r} \leftarrow \mathbb{Z}_m^d$ and send server $(b_1, \dots, b_t)$ the point $\mathbf{r} + \mathsf{CRT}(b_1, \dots, b_t) \cdot \mathbf{u}_{i^*} \in \mathbb{Z}_m^d$.
- Given a vector $\mathsf{qu} \in \mathbb{Z}_m^d$, server $(b_1, \dots, b_t)$ will reply with:

$$\mathsf{ans}_{b_1,\dots,b_t} = \sum_{i \in [N]} \mathsf{DB}_i \cdot \prod_{j=1}^{t} g_j^{\langle \mathsf{qu}, \mathbf{v}_i \rangle} = \sum_{i \in [N]} \mathsf{DB}_i \cdot \prod_{j=1}^{t} g_j^{\langle \mathbf{r} + b_j \mathbf{u}_{i^*}, \mathbf{v}_i \rangle}.$$

- The client will now compute:

$$\begin{aligned}
\sum_{b_1,\dots,b_t \in \{0,1\}} (-1)^{\sum_{i \in [t]} b_i} \mathsf{ans}_{b_1,\dots,b_t} &= \sum_{b_1,\dots,b_t \in \{0,1\}} (-1)^{\sum_{i \in [t]} b_i} \sum_{i \in [N]} \mathsf{DB}_i \cdot \prod_{j=1}^{t} g_j^{\langle \mathbf{r} + b_j \mathbf{u}_{i^*}, \mathbf{v}_i \rangle} \\
&= \sum_{i \in [N]} \mathsf{DB}_i \cdot \left[\prod_{j=1}^{t} \left(\sum_{b_j \in \{0,1\}} (-1)^{b_j} g_j^{\langle \mathbf{r} + b_j \mathbf{u}_{i^*}, \mathbf{v}_i \rangle} \right) \right] \\
&= \sum_{i \in [N]} \mathsf{DB}_i \cdot \left[\prod_{j=1}^{t} g_j^{\langle \mathbf{r}, \mathbf{v}_i \rangle} \left(1 - g_j^{\langle \mathbf{u}_{i^*}, \mathbf{v}_i \rangle} \right) \right] \\
&= \mathsf{DB}_{i^*} \cdot \prod_{j=1}^{t} g_j^{\langle \mathbf{r}, \mathbf{v}_{i^*} \rangle} (1 - g_j),
\end{aligned}$$

  from which they can recover DB_{i^*}.

When adapting this to tree evaluation, we face the following natural obstacles:

1. The CIR protocol needs to be composable with itself in order to recursively apply it when going up the tree. To this end, we ensure that our queries and reconstructed answers both take the form of adding a matching vector into a catalytic register.
2. Thus, when carrying out one level of tree evaluation, we assume we can update $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{u}_a$ and $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{u}_b$. However, what we really need for CIR is to be able to make queries that are indexed by the tuple (a, b). This can be seen in the equation in Definition 4.1. Our solution is roughly inspired by the fact that the tensored collection of vectors $\{\mathbf{u}_a \otimes \mathbf{u}_b, \mathbf{v}_a \otimes \mathbf{v}_b \ : \ a, b \in \{0,1\}^\ell\}$ is itself a matching vector family over $\mathbb{Z}_m^{d^2}$. We cannot actually write these tensor products down, but instead stream through them to remain in low space.

3. The recursive composability requires the CIR queries and answers to have the same type. For the simple scheme sketched above, this cannot be true! DB_{i^*} lives in $\mathbb{Z}_q$, where there needs to be an element of multiplicative order m, while the queries live in $\mathbb{Z}_m$. To remedy this, we move away from $\mathbb{Z}_q$ to a formal polynomial ring over $\mathbb{Z}_m$—following the original presentations of [DGY11, Efr12, DG16, GKS25]—where we can adjoin formal variables of multiplicative degree dividing m. As noted in Remark 3.12, this leads us to work over the ring $\mathbb{Z}_m[X_1, \ldots, X_t]/(X_1^{p_1} - 1, \ldots, X_t^{p_t} - 1)$. Following this approach comes with minor difficulties, but we can simplify the resulting construction to get rid of the polynomial ring, yielding the construction in Section 3.1.

5 Application: New Time-Space-Catalytic Space Tradeoffs

5.1 The Reduction of Williams

We recall the reduction from $\mathsf{Time}[t]$ to tree evaluation.

Theorem 5.1 ([Wil25]). *For every language L in $\mathsf{Time}[t]$, there is a machine that on input $x \in \{0,1\}^n$ runs in space $O(\sqrt{t})$ and outputs a $\mathsf{TreeEval}_{h,\ell}$ instance with $h = O(\sqrt{t})$ and $\ell = O(\sqrt{t})$ such that the output of the tree eval instance is $L(x)$.*

We remark that the result of Williams outputs a $\mathsf{TreeEval}$ instance with fanin $r \geq 2$ for some constant r that depends on the language L, but the result as stated above is immediate from Lemma 2.7.

Subsequently, Shalunov gave a direct reduction from size-S circuit evaluation to $\mathsf{TreeEval}_{h,\ell}$:

Theorem 5.2 ([Sha25]). *There is a $O(\sqrt{S})$ space algorithm that, given a circuit C with S gates[7] and input $x \in \{0,1\}^n$, outputs a $\mathsf{TreeEval}_{h,\ell}$ instance with $h = O(\sqrt{S})$ and $\ell = O(\sqrt{S})$ such that the output of the tree eval instance is $C(x)$.*

5.2 Applications of Catalytic Tree Evaluation

Corollary 1.3 follows immediately from Corollary 3.2 and Theorem 5.1. We can also plug in Corollary 3.3 to obtain a different corollary. For this, we use that the algorithm of Theorem 5.1 can produce a $\mathsf{TreeEval}$ instance of height t/b and $\ell = b$ for any space constructible function b. We instantiate it with $b = \sqrt{t} \cdot \log^{1/4}(t)$ and obtain the following:

Corollary 5.3. $\mathsf{Time}(t) \subseteq \mathsf{CatTimeSpace}\left[\exp\exp(O(\sqrt{\log t})), O(\sqrt{t} \cdot \log^{1/4}(t) \log\log\log t), 2^{O(\sqrt{t}\cdot\log^{1/4}(t))}\right]$.

Finally, combining Theorem 5.2 with Theorem 1.1 immediately gives the following:

Corollary 5.4. For any $\varepsilon > 0$, size-S circuit evaluation can be decided in $O(\sqrt{S})$ free space, $2^{O(S^\varepsilon)}$ catalytic space, and $2^{O(\sqrt{S})}$ time.

Acknowledgements

A.H. and S.R. thank Vinod Vaikuntanathan for encouraging this work and for helpful discussions. E.P. thanks James Cook, Ian Mertz, and Ryan Williams for useful discussions.

[7] The description size is thus $O(S \log S)$ bits.

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. 9

[AM25] Aryan Agarwala and Ian Mertz. Bipartite matching is in catalytic logspace. *CoRR*, abs/2504.09991, 2025. 5

[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. 5

[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. 5, 7

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992. 5

[BCK+14] Harry Buhrman, Richard Cleve, Michal Kouckỳ, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *STOC*, 2014. 1, 2, 5, 6

[BDL14] Abhishek Bhowmick, Zeev Dvir, and Shachar Lovett. New bounds for matching vector families. *SIAM J. Comput.*, 43(5):1654–1683, 2014. 9

[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. 3, 5

[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. 5

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50. IEEE Computer Society, 1995. 3, 4

[CGKS98] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *J. ACM*, 1998. 3

[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. 5

[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. 5

[CM24] James Cook and Ian Mertz. Tree evaluation is in space $O(\log n \cdot \log \log n)$. In *STOC*, 2024. 1, 3, 4, 5, 6, 9, 17

[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, 2, 5

[CP25] James Cook and Edward Pyne. Efficient catalytic graph algorithms. *CoRR*, abs/2509.06209, 2025. 5, 6, 7

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server PIR with subpolynomial communication. *J. ACM*, 2016. 2, 4, 5, 7, 12, 18, 19

[DGY11] Zeev Dvir, Parikshit Gopalan, and Sergey Yekhanin. Matching vector codes. *SIAM Journal on Computing*, 40(4):1154–1178, 2011. 2, 5, 7, 8, 12, 18, 19, 22

[Efr09] Klim Efremenko. 3-query locally decodable codes of subexponential length. In *STOC*, 2009. 5

[Efr12] Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 2012. 5, 7, 18, 19

[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. 5

[GGMT25] William Timothy Gowers, Ben Green, Freddie Manners, and Terence Tao. On a conjecture of marton. *Annals of Mathematics*, 201(2):515–549, 2025. 9

[GKS25] Fatemeh Ghasemi, Swastik Kopparty, and Madhu Sudan. Improved PIR schemes using matching vectors and derivatives. In *STOC*, 2025. 2, 4, 5, 12, 18, 19

[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. 2, 4, 6

[Gop06] Parikshit Gopalan. *Computing with Polynomials over Composites.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2006. 23

[Gro00] Vince Grolmusz. Superpolynomial size set-systems with restricted intersections mod 6 and explicit Ramsey graphs. *Comb.*, 20(1):71–86, 2000. 2, 5, 7, 8, 24

[HPV77] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM (JACM)*, 24(2), 1977. 5

[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. 5

[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. 5

[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. 5

[LBA25] Or Lasri, Amos Beimel, and Bar Alon. Simplified PIR and CDS protocols and improved linear secret-sharing schemes. In *TCC*. Springer, 2025. 5, 18

[Liu13] David Liu. Pebbling arguments for tree evaluation. *CoRR*, abs/1311.0293, 2013. 5

[Ruz99] Imre Ruzsa. An analog of Freiman's theorem in groups. *Astérisque*, 258(199):323–326, 1999. 9

[Sha25] Yakov Shalunov. Improved bounds on the space complexity of circuit evaluation. *Electron. Colloquium Comput. Complex.*, TR25-078, 2025. 19

[Sto23] Manuel Stoeckl. Private communication, 2023. 6

[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. 9

[Wil25] R Ryan Williams. Simulating time with square-root space. In *STOC*, 2025. 1, 2, 5, 19

[WY05] David Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *CCC*. IEEE, 2005. 3, 4

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 2008. 5

[Yek12] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3), 2012. 5

A Verifying the Uniformity of the Matching-Vector Family

The fact that the matching-vector family of [DGY11, Lemma 11] is logspace uniform is not explicitly stated, but follows immediately from the construction. We verify this below, making no claims to originality. We exactly follow their notation.

Definition A.1. Let $p_1, \ldots, p_t$ be distinct primes and let $m = \prod_i p_i$. The canonical set S in $\mathbb{Z}_m$ is the set of nonzero s where $s \in \{0, 1\} \mod p_i$ for every i.

Lemma A.2. *Let $m = \prod_{i=1}^{t} p_i$ be a product of distinct primes that are given as input. Let w, g and $e_1, \ldots, e_t$ be given such that $\prod_{i=1}^{t} p_i^{e_i} > w$ and $h \geq w$. Let $d = \max_i p_i^{e_i}$. There exists a space $O(\log N + d \log h + \log m)$-uniform[8] matching vector family of size $N = \binom{h}{w}$ in $\mathbb{Z}_m^n$ where $n = \binom{h}{\leq d}$.*

We first define polynomial matching families:

[8] In both regimes we work with $\log(N) \geq d \log h$, so this meets the definition of logspace-uniformity.

Definition A.3 (Polynomial Matching Family, Definition 35)**.** Let $S \subseteq \mathbb{Z}_m \setminus \{0\}$. We say that a set of polynomials $\mathcal{F} = \{f_1, \ldots, f_k\} \subseteq \mathbb{Z}_m[z_1, \ldots, z_h]$ and a set of points $x = \{x_1, \ldots, x_k\} \subseteq \mathbb{Z}_m^h$ form a space-s uniform polynomial S-matching family of size k if

- for all $i \in [k]$, $f_i(x_i) = 0$;
- for all $i, j \in [k]$ such that $i \neq j$, $f_j(\mathbf{x}_i) \in S$.
- There is a space s algorithm that prints $\mathcal{F}$.

First, such a family can be turned into matching vectors by an observation of Sudan. Let $\mathcal{F}, \mathcal{X}$ be a logspace-uniform k-sized polynomial matching family. For $i \in [k]$, let $\mathsf{supp}(f_i)$ denote the set of monomials in the support of the polynomial f_i. Define $\mathsf{supp}(\mathcal{F}) = \bigcup_{i=1}^{k} \mathsf{supp}(f_i)$ and $\dim(\mathcal{F}) = |\mathsf{supp}(\mathcal{F})|$.

Lemma A.4 (Lemma 36)**.** *A space-s uniform k-size polynomial S-matching family $\mathcal{F}, \mathcal{X}$ over $\mathbb{Z}_m$ yields a space-$O(s + \log m)$ uniform k-sized matching vector family in $\mathbb{Z}_m^n$, where $n = \dim(\mathcal{F})$.*

Proof. We have by assumption that we can enumerate over the monomials in $\mathcal{F}$ in space $O(s)$ (and hence print the coefficient of each monomial).

Let $\textsc{mon}_1, \ldots, \textsc{mon}_n$ be these monomials, and let

$$f_j = \sum_{l=1}^{n} c_{jl} \cdot \textsc{mon}_l$$

where $c_{jl} \in \mathbb{Z}_m$.

Finally, we let $\mathbf{u}_i \in \mathbb{Z}_m^n$ be the n-dimensional vector of the coefficients of f_i. It is straightforward that we can enumerate over monomials in $O(s)$ and determine if f_i contains this monomial, and if so read off the coefficient. Next, let $\mathbf{v}_j \in \mathbb{Z}_m^n$ be the vector of evaluations of the monomials at x_j. Here we can again enumerate over the vectors x_j in space $O(s)$ and perform this evaluation in space $O(s + \log m)$. □

We then construct such a family. We require low-degree polynomials which compute the weight of x mod $p_i^{e_i}$:

Lemma A.5 (Theorem 2.16 [Gop06])**.** *There is a space $O(d \log h)$ algorithm[9] that given $i \in [t]$ and w prints an explicit multilinear polynomial $f_i(z_1, \ldots, z_h) \in \mathbb{Z}_{p_i}[z_1, \ldots, z_h]$ where $\deg(f_i) \leq p_i^{e_i} - 1$ and for $x \in \{0,1\}^h$:*

$$f_i(x) = \begin{cases} 0 \mod p_i & \sum_i x_i \equiv w \mod p_i^{e_i} \\ 1 & \text{o.w.} \end{cases}$$

From this we immediately obtain that in space $O(d \log h + \log m)$ we can obtain the following:

Corollary A.6 (Corollary 38)**.** There is a space $O(d \log h + \log m)$ algorithm that prints an explicit degree d multilinear polynomial $f_i(z_1, \ldots, z_h) \in \mathbb{Z}_m[z_1, \ldots, z_h]$ for $x \in \{0,1\}^h$:

$$f_i(x) = \begin{cases} 0 \mod m & \sum_i x_i = w \\ s \mod m & \sum_i x_i < w \end{cases}$$

In the above, $\sum_i x_i$ is being computed over $\mathbb{Z}$.

[9]The explicitness is not stated but is immediate from the construction.

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

Proof. 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 $O(\log N)$ as follows. We let $S = 0$ and $i = w$ and choose c_i maximal such that

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

Since we can store S using $O(\log N)$ bits since all values are bounded by N (so we can obviously compute coefficients in space $O(\log N)$) we are done. □

Proof of Lemma 11. We construct a polynomial S-matching family and apply Lemma A.4.

We work with subsets $T \subseteq [h]$ of size w. We use the $O(\log N)$-space function $[N] \to \{0,1\}^T$ of Theorem A.7 and hence index these sets as numbers in $[N]$ without loss of generality.

For each such set T, letting f be the polynomial of Corollary A.6, we define f_T as the polynomial where we set $z_j = 0$ for $j \notin T$. We can clearly construct this polynomial in space $O(\log k + d \log h + \log m)$. Finally, let $x_T \in \{0,1\}^h$ be the indicator of T. We WLOG extend $\mathsf{supp}(\mathcal{F})$ to be all monomials of degree at most d, which we can enumerate over in space $O(d \log h)$. Thus, the total space required to print the polynomial family (and hence the matching vector family) is $O(\log k + (d \log h) + \log m)$ as desired. □

B Proof of Corollary 2.13

Item 1 is immediate and exactly the result proven by [Gro00] (by taking $w = \Theta(\ell/\log \ell)$). Item 2 also follows directly from Theorem 2.12 by taking $w = \Theta(\ell/\sqrt{\log \ell})$ and $t = \sqrt{\log w}$. We know by the prime number theorem that $m = t^{t+o(t)}$. The dimension can be bounded above by $(mw)^{1/t} + 1$ times $\binom{\lceil w^{1+1/t} \rceil}{\lfloor (mw)^{1/t} \rfloor}$, noting that the last binomial coefficient must be the largest since $t^{1+o(1)} < w/2 \Rightarrow m^{1/t} < w/2 \Rightarrow (mw)^{1/t} < w^{1+1/t}/2$. To bound the first factor (the number of binomial coefficients), note that:

$$\begin{aligned}
(mw)^{1/t} &\leq t^{1+o(1)} w^{1/t} \\
&= (\log w)^{1/2+o(1)} 2^{\sqrt{\log w}} \\
&\leq \exp(O(\sqrt{\log w})) \\
&= \exp(O(\sqrt{\log \ell})).
\end{aligned}$$

We can bound the largest binomial coefficient by:

$$\begin{aligned}
\left(\frac{e\lceil w^{1+1/t} \rceil}{\lfloor (mw)^{1/t} \rfloor}\right)^{(mw)^{1/t}} &\leq \left(\frac{3w^{1+1/t}}{(mw)^{1/t}}\right)^{(mw)^{1/t}} \\
&\leq (w/t^{1-o(1)})^{t^{1+o(1)} w^{1/t}} \\
&= \exp\left(t^{1+o(1)} \cdot w^{1/t} \cdot (\log w - \log t + o(\log t))\right)
\end{aligned}$$

$$= \exp\left((\log w)^{1/2+o(1)} \cdot 2^{\sqrt{\log w}} \cdot \log w\right)$$
$$= \exp(\exp(O(\sqrt{\log w})))$$
$$= \exp(\exp(O(\sqrt{\log \ell}))).$$

Multiplying these two bounds implies the conclusion. □

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il