Electronic Colloquium on Computational Complexity, Report No. 90 (2025)

Linear Prover IOPs in Log Star Rounds

Noor Athamnah* Noga Ron-Zewi† Ron D. Rothblum‡

July 10, 2025

Abstract

Interactive Oracle Proofs (IOPs) form the backbone of some of the most efficient general-purpose cryptographic proof-systems. In an IOP, the prover can interact with the verifier over multiple rounds, where in each round the prover sends a long message, from which the verifier only queries a few symbols.

State-of-the-art IOPs achieve a linear-size prover and a poly-logarithmic verifier but require a relatively large, logarithmic, number of rounds. While Fiat-Shamir heuristic can be used to eliminate the need for actual interaction, in modern highly-parallelizable computer architectures such as GPUs, the large number of rounds still translates into a major bottleneck for the prover, since it needs to alternate between computing the IOP messages and the Fiat-Shamir hashes. Motivated by this fact, in this work we study the round complexity of linear-prover IOPs.

Our main result is an IOP for a large class of Boolean circuits, with only $O(\log^*(S))$ rounds, where $\log^*$ denotes the iterated logarithm function (and S is the circuit size). The prover has linear size $O(S)$ and the verifier runs in time $\text{polylog}(S)$ and has query complexity $O(\log^*(S))$. The protocol is both conceptually simpler, and strictly more efficient, than prior linear prover IOPs for Boolean circuits.

*Technion. Email: noor.athamnah@gmail.com

†University of Haifa. Email: noga@cs.haifa.ac.il

‡Succinct. Email: rothblum@gmail.com

ISSN 1433-8092

Contents

1 Introduction

Succinct arguments enable a prover to efficiently prove correctness of a complex computational statement to a weak verifier. Such proof-systems, which originated and have been studied for decades in the theory literature, are now starting to enjoy broad interest and adoption also in practice.

In this work we study proof-systems for proving correctness of computations expressed in arguably the most basic computational model: Boolean circuits. Thus, we consider a fixed Boolean circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ (consisting of gates of fan-in 2). Both parties are given as input $x \in \{0,1\}^n$, and the prover should convince the verifier that there exists some witness $w \in \{0,1\}^m$ such that $C(x,w) = 1$. The witness w is given as an auxiliary input to the prover, which should be implemented as a (small as possible) Boolean circuit.

The motivation for studying the Boolean circuit model is two-fold:

1. **(Natural and Basic Model:)** The Boolean circuit model is a natural and extremely basic model of computation. It allows for a clear apples-to-apples comparison between the cost of computing vs. the cost of proving. There is no question about the types of operations that are allowed and no restriction to arithmetic computations (in contrast to the arithmetic circuit model). The model is also not tied down to the structure imposed by the underlying proof-system.

2. **(Batching via Bit-Slicing:)** Suppose that a function f can be proved by a Boolean circuit of size S. Observe that on a RAM machine with a word size of w bits, we can use this proof-system to prove correctness of w instances of f using only S word operations via *bit-slicing*: that is, we pack all of the i-th input bits of the w copies into a single machine word and then emulate the bit operation of the circuit (such as AND and XOR) by running the corresponding word operations, while observing that the word operations are actually operating on the individual bits within each word according to the structure of the Boolean circuit.

The state-of-the-art proof-systems in the Boolean circuit model, due to Ron-Zewi and Rothblum [RR25] and Holmgren and Rothblum [HR22], construct a prover whose size (as a Boolean circuit) is strictly linear in the original computation.

In this work we focus on the *number of rounds of interaction* in the proof-system. Part of our motivation stems from the fact that the number of rounds is a fundamental resource in proof-systems and so we should try to minimize it. While it is true that these rounds can potentially be eliminated via Fiat-Shamir [FS86], this heuristic is known not to be sound in general [Bar01, GK03, BBH+19], and there are attacks even against some practical protocols [KRS25].

However, even putting these considerations aside, a large part of our motivation actually comes from *efficiency*. Many modern proof-systems are currently being implemented on GPUs (see, e.g., [Suc]). In this extremely parallelizable architecture, the number of rounds turns out to be a fundamental bottleneck. Even when employing the Fiat-Shamir transform, if the underlying interactive protocol has t-rounds, the prover has to run t sequential steps (of computing the next round message and applying the Fiat-Shamir hash function). Thus, reducing the number of rounds almost immediately improves the number of parallel steps (aka latency) of the proof-system.

Essentially all succinct arguments in the literature are built by first constructing an information-theoretic proof-system, and then applying a cryptographic compiler. We also follow this approach

and so focus on the underlying information-theoretic component, which in our case is an *interactive oracle proof* (IOP). Recall that the IOP model [BCS16, RRR21] is an extension of the PCP model: it is an interactive protocol in which the prover is allowed to send long strings (aka oracles) in each round, so that the verifier only reads a few bits from each string.

Most of the practical IOPs in the literature, including those of [RR25, HR22] have a number of rounds that is logarithmic in the circuit size. Some exceptions are interleaving based IOPs such as [AHIV17, BCG+17, GLS+23, BFK+24] which are constant-round, but require $O(\sqrt{S})$ verification and have a super-linear prover for Boolean computations. Another IOP by Arnon *et al.* [ACY23] has $O(\log\log(S))$ rounds but has linear verification, operates over a large finite field, and with a super-linear size prover (but offers a better soundness vs. alphabet size tradeoff).

We know that in principle it is possible to construct IOPs which have only a *single* round — indeed, these are PCPs. Alas, these single round IOPs seem far less efficient than their interactive counterparts. Indeed, it is a major open question to construct a PCP of linear length (let alone with a linear-size prover). Other IOPs such as that of [RR24, RW24] have a constant number of rounds and linear length, but the prover runtime is a large polynomial.

In this work we ask whether there is a way to balance between these results: obtain highly efficient provers for Boolean computations, while using significantly less rounds. Thus, the main question that we study in this work is: what is the smallest number of rounds with which we can prove correctness using a strictly linear-size Boolean circuit?

1.1 Our results

Our main result is a construction of an IOP, for satisfiability of a large class of Boolean circuits, with a linear-size prover and only $O(\log^*(S))$ rounds of interaction, where S denotes the circuit size. Recall that the iterated logarithm function $\log^*$ is the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1. The function $\log^*(n)$ grows *very* slowly, and in particular is less than or equal to 5, for $n \leq 2^{65536}$ (which, for context, is roughly 20,000 orders of magnitude larger than the number of atoms in the universe).

Similarly to [RR25] we focus on the constant soundness error regime, but note that via a technique from [HR22], the soundness error can be reduced to $2^{-\lambda}$ while only paying a (multiplicative) $\text{polylog}(\lambda)$ overhead (see Remark 1.3 below).

Theorem 1.1 (Informally Stated, see Section 5)**.** *Let $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be a fan-in 2 "regular" Boolean circuit (with constant fan-in) of size S . Then, there exists an $O(\log^*(S))$-round IOP for the language $\{x \in \{0,1\}^n : \exists w \in \{0,1\}^m,\ s.t.\ C(x,w) = 1\}$, with a constant soundness error, where the prover can be implemented as a size $O(S)$ Boolean circuit. Following a preprocessing step, the verifier has size $\text{polylog}(S)$ and query complexity $O(\log^*(S))$.*

Theorem 1.1 is a strict improvement over the prior state-of-the-art for proof-systems for Boolean circuits, which achieve similar parameters but with $O(\log(S))$ rounds, and a corresponding number of queries. We elaborate on the precise technical meaning of "regular circuit" in Section 5, but loosely speaking we mean that the wiring pattern has a lot of repeated structure. As particular special cases of interest, this includes batch verification circuits of the form $C(x_1, \ldots, x_k; w) = \bigwedge_i C_0(x_i; w)$, or sequential composition of a single circuit $C(x) = C_0(C_0(\ldots C_0(x))$, where in both cases C_0 is a smaller base circuit of size at most $S/\text{polylog}(S)$, which can be arbitrary. The cost of the preprocessing step is quasi-linear in the "irregularity" of the circuit (e.g., the size of C_0).[1]

[1] Our focus is on the setting of sub-linear verification following a public pre-processing step. We remark that if

While our focus is on Boolean circuits, our techniques extend directly to arithmetic circuits over large fields. To the best of our knowledge, all prior results over such large fields, with a linear-size prover, used at least a logarithmic number of rounds. We remark that the extension of our result to arithmetic circuits over large fields can work for arbitrary (rather than just regular) arithmetic circuits (using an arithmetization as in [BCG20]).

Remark 1.2. *Given how slowly* $\log^*$ *grows, it is interesting to consider also the hidden constant in the big O notation in the round complexity of Theorem 1.1. Our somewhat loose analysis shows that the number of rounds is bounded by* $2\log^*(S) + O(1)$ *(see Claim 4.4), but we believe that a tighter analysis of the same protocol should yield a round complexity of* $(1 + o(1)) \cdot \log^*(S) + O(1)$.

As an additional contribution, we also construct a new IOP for Boolean circuits with a linear-size prover, and logarithmically many rounds. This matches the prior works of [RR25, HR22] but via a conceptually simpler protocol. The new protocol also achieves better communication complexity and uses the code-switching technique of [RR24] in a more lightweight manner.

We emphasize that our focus is theoretical in nature and we do not claim these protocols, as described, to be practical. Nevertheless, similarly to prior related results in the literature (e.g., code-switching [RR24]) we believe that similar techniques can be used to reduce rounds also in practice.

Remark 1.3. *Our protocols have a constant soundness error, which can be reduced to* $2^{-\lambda}$ *via parallel repetition, or more efficiently (i.e., with only* $\mathrm{polylog}(\lambda)$ *multiplicative overhead) via a technique from [HR22].*

However, if the Fiat-Shamir transform is to be employed, the protocol needs a stronger notion of soundness called round-by-round soundness *[CCH+19]. Unfortunately, at first glance, both a* λ*-fold parallel repetition, as well as the method of [HR22], seem to yield round-by-round soundness error of* $2^{-\Omega(\lambda/\ell)}$ *(see [CCH+18, Corollary 5.7]), where* ℓ *is the round complexity of the protocol (rather than the desired* $2^{-\Omega(\lambda)}$*).*

Looking more closely though, we observe that our base protocol has round-by-round soundness ε_i *in the* i*-th round, where the* ε_i*'s are gradually shrinking, and in particular* $\sum_i \varepsilon_i \ll 1$*. This implies that the* λ*-fold parallel repetition of our protocol, as well as the more efficient amplification via [HR22], reduces the round-by-round soundness error to* $2^{-\Omega(\lambda)}$.

1.2 Technical overview

Our construction of a round-efficient IOP for circuit satisfiability follows the basic blueprint in [RR25, HR24]. In particular, the key technical ingredient in [RR25] is a protocol in which the verifier gets oracle access to encodings, under a good error-correcting code, of two strings x and y, and needs to verify their inner product $\langle x, y\rangle = \sum_i x_i \cdot y_i \pmod 2$, using an IOP, while making only a few queries to both the proof oracles and the (encoded) inputs.[2] We refer to such a protocol as an "inner-product IOP".[3]

---

one allows linear-time verification, or *private* (and unreusable) pre-processing, with poly-logarithmic verification, our IOP can be extended to any circuit (rather than just regular ones) similarly to [RR25].

[2] In the technical sections we consider a more general case in which the input is $x_1, \ldots, x_d$ and the goal is to verify $\sum_i x_1(i) \cdots x_d(i)$. In the technical overview, for sake of simplicity, we focus on the case $d = 2$.

[3] [RR25] uses the terminology of "multi-sumcheck" but in retrospect we find the term "inner product check" more appealing.

Definition 1.4 (Inner-product IOP, Informally Stated)**.** *An* inner product IOP relative to a code $E : \{0,1\}^n \to \{0,1\}^{n'}$*, is an IOP in which the prover gets as input* $x, y \in \{0,1\}^n$ *and the verifier gets oracle access to* $E(x)$ *and* $E(y)$ *and explicit access to a value* $v \in \{0,1\}$*. If* $\langle x, y \rangle = v$*, the verifier should accept when interacting with the honest prover, and if* $\langle x, y \rangle \neq v$ *the verifier should reject whp when interacting with* any *cheating prover.*

Loosely speaking, the inner product IOP can be used (more or less) as a drop-in replacement for a standard sumcheck used in traditional arithmetization.[4]

The traditional solution to this problem is to use a code E that is *multiplicative* — that is, a code in which the point-wise product between two codewords is guaranteed to reside in a related linear code. Using such a code, one can just run a "sumcheck-like" procedure on the product codeword $E(x) \circ E(y)$. Indeed, in the special case that the code E is the multilinear extension (which is multiplicative), the classical sumcheck protocol [LFKN92] can be directly used as an inner product check.

The benefit of looking at the generalized notion is that we can now consider inner product IOPs, relative to more efficient codes — in particular, *linear-time encodable codes* (e.g., [Spi96]). The challenge that we face however is that there are no known linear-time encodable multiplications codes.

The key technical contribution of our work is a new inner product IOP, relative to a linear-time encodable error-correcting code E, with a linear-size prover and only $O(\log^*(n))$ rounds of interaction (and a constant soundness error, which can later be reduced via Remark 1.3).

1.2.1 Warmup: Logarithmic round IOP

As a warmup, we first consider the goal of constructing an inner product IOP with a logarithmic number of rounds and a prover implemented as a linear-size Boolean circuit. While this goal is non-trivial, it was already achieved in [RR25]. However, we give here a conceptually simpler protocol which we find interesting in its own right, and will also be very helpful in our eventual goal of $O(\log^*(n))$ rounds. Furthermore, while the protocol of [RR25] is an IOP (i.e., that involves sending large linear-length oracle messages), the logarithmic round protocol that we propose here can be viewed as a standard interactive proof, with communication n^ε, for any desired $\varepsilon > 0$ (but can also be transformed into an IOP with $O(\log n)$ query complexity). See Remark 1.7 below for a more detailed comparison with the [RR25] protocol.

For the warmup, our main conceptual insight is that the inner product protocol can be broken down into two distinct steps:

1. As our first step, we construct an inner product protocol with a linear-size prover, but assume that the verifier can access the *multilinear extensions* (MLEs) of the inputs x and y, rather than their encoding under the linear-time encodable code E. Furthermore, we ensure that the queries to the multilinear extension have a particular "nice" structure, on which we will elaborate shortly.

2. The second step shows that if the verifier is given oracle access to an encoded message $E(x)$, a linear-size prover can prove to it such nice claims about the multilinear extension $\hat{x}$ of x.

[4] For the arithmetization we also need the code to be locally testable and (relaxed) locally correctable (via auxiliary IOPs). This is usually easy to achieve via code tensoring [Vid15, GRR18].

Given these two components, an inner product protocol for E follows easily: the parties run the first step to obtain the "nice" claims about $\hat{x}$ and $\hat{y}$, and then use the second step to verify these claims (separately) wrt to $E(x)$ and $E(y)$.

We proceed to elaborate on these two steps.

Linear-size inner product IOP relative to MLE. Our goal now is to construct an inner product protocol, in which the prover gets as input $x, y \in \{0,1\}^n$ and is trying to prove that $\langle x, y \rangle = v$, for some $v \in \{0,1\}$. We view the verifier as receiving only v as input, and outputting claims about the multilinear extensions of x and y (rather than accepting/rejecting). Thus, the protocol should be thought of as a reduction from the inner product problem to claims about the MLEs of the two strings. We emphasize that this protocol does not involve the linear-time code E (which will only come into play in the second step).

Observe that the classical sumcheck protocol solves *exactly* this problem. To see this, it will be useful to make a syntactic change of viewing the inputs as functions rather than strings. Namely, let $f, g : \{0,1\}^m \to \{0,1\}$ be functions whose truth tables are exactly the strings x and y, respectively, and where $m = \log(n)$. Let $\hat{f}, \hat{g} : \mathbb{F}^m \to \mathbb{F}$ be the respective multilinear extensions of f and g.[5]

Thus, we can solve our problem by just running the sumcheck protocol to verify that $v = \sum_{b \in \{0,1\}^m} \hat{f}(b) \cdot \hat{g}(b)$ (indeed, this uses the fact that the MLE is a multiplication code). The sumcheck prover can be implemented as a linear-size arithmetic circuit over the field $\mathbb{F}$. The problem however is that the sumcheck protocol is an m-round protocol, with a soundness error of $O(1/|\mathbb{F}|)$ in each round. Thus, to get a meaningful bound, the size of the field $|\mathbb{F}|$ must be larger than m. Since we want the prover to be a *Boolean* circuit, we need to emulate each of the field operations via a Boolean circuit. Unfortunately, this yields a super-linear size prover.

To explain our approach, let us zoom in on the computation that the prover does to produce its first message. Recall that the first message in the sumcheck protocol is (a concise description of) the polynomial $p_1(\lambda) = \sum_{b \in \{0,1\}^{m-1}} \hat{f}(\lambda, b) \cdot \hat{g}(\lambda, b)$. Using the formula directly, it is not difficult to construct a linear-size arithmetic circuit that computes p_1. Indeed, if the field were constant-sized, this would suffice for us. But, as discussed above, a constant-sized field leads to a constant soundness error. While we could tolerate this in the first round, if we apply the same logic to the subsequent $m - 1$ rounds then the soundness error becomes extremely close to 1.

Our main observation, which is related to the main observation in [RR25] (and later used also in [ACFY24, ACFY25]) is that as the protocol progresses, the size of the instance being proved shrinks, and so we can afford to invest more time (relative to the current instance size). In particular, we can use a larger field.

Thus, rather then working with a single finite field, we will be working with a tower of extension fields $\mathbb{F}_1 \subset \mathbb{F}_2 \subset \cdots \subset \mathbb{F}_m$ (whose exact sizes will be determined soon). In round i of the sumcheck, the prover message is computed over the field $\mathbb{F}_i$, but the verifier samples a random challenge element r_i from the extension field $\mathbb{F}_{i+1} \supset \mathbb{F}_i$. Since each sumcheck round halves the problem size, and the cost of emulating field operations by a Boolean circuit, for a size $|\mathbb{F}_i|$ field, is $\text{polylog}(|\mathbb{F}_i|)$, the overall prover computation can be bounded by:

$$\sum_{i=1}^{m} 2^{m-i} \cdot \text{polylog}(|\mathbb{F}_i|),$$

[5] Recall that the multilinear extension $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ of a function $f : \{0,1\}^m \to \mathbb{F}$ is the unique multilinear polynomial that agrees with f on $\{0,1\}^m$, see Fact 2.8 for details.

whereas the soundness error (over all rounds) is proportional to:

$$\sum_{i=1}^{m} \frac{1}{|\mathbb{F}_{i+1}|}.$$

By setting $|\mathbb{F}_i| \approx 2^i$ we get the best of both worlds — a strictly linear-size prover (as a Boolean circuit) and a constant soundness error.

Remark 1.5. *It is worth nothing that a vast range of different parameters for the field sizes would have also worked here (ranging from polynomial in i to almost doubly-exponential), but eventually lead to a similar asymptotic result. We will leverage this fact later on when reducing the number of rounds.*

Thus, we run the sumcheck protocol, while gradually increasing the field size as the protocol progresses. At the end of the sumcheck, the verifier will need to verify a claim of the form $\hat{f}(r) \cdot \hat{g}(r) = v'$, for $r = (r_1, \ldots, r_m) \in \mathbb{F}^m$ and $v' \in \mathbb{F}$, where $\mathbb{F} = \mathbb{F}_m$ is the field used in the last round of the protocol (i.e., the largest extension field). To verify this claim, the verifier asks the prover to provide its claimed values for both $\hat{f}(r)$ and $\hat{g}(r)$ and check that their product is equal to v'.

This simple tweak of sumcheck, in which we gradually increase the field sizes, suffices for our first goal — reducing the inner product claim to separate claims about the MLEs of f and of g.

Verifying MLE Queries via Tensors. The task at hand now is the following. The prover gets as input $f : \{0,1\}^m \to \{0,1\}$ and $r \in \mathbb{F}^m$. The verifier get oracle access to an encoding of the truth table of f under a linear-time encodable code, as well as the point r and a scalar $v \in \mathbb{F}$, and needs to be able to verify that $\hat{f}(r) = v$.

Before discussing how to prove this claim, let us first consider the task of merely *computing* it (as, needless to say, proving can be no easier than computing). That is, how can the prover even compute the value $\hat{f}(r)$. By definition of the multilinear extension, this value is equal to:

$$\hat{f}(r) = \sum_{b \in \{0,1\}^m} eq(r, b) \cdot f(b), \tag{1}$$

where $eq(r, b) = \prod_{i=1}^{m} eq_1(r_i, b_i)$ and $eq_1(\alpha, \beta) = \alpha \cdot \beta + (1-\alpha) \cdot (1-\beta)$ (see Section 2.2 for additional details). The known (efficient) methods for computing this expression boil down to generating the sequence of values $(eq(r, b))_{b \in \{0,1\}^m}$ (see [VSBW13]). Unfortunately, generating this sequence *requires* a Boolean circuit of size at least $2^m \cdot \log(|\mathbb{F}|)$, since that is the amount of space required to just store this sequence. In our case the field $\mathbb{F}$ will be the last field from the previous step, which has a super-constant size and therefore it seems that even just computing $\hat{f}(r)$ inherently requires a super-linear size circuit.

Notice however that the point $r \in \mathbb{F}^m$ generated by the previous protocol is not actually arbitrary. Specifically, in the i-th round of our sumcheck variant, the challenge r_i was chosen randomly from the field $\mathbb{F}_i$. Thus, the point r actually belongs to $\mathbb{F}_1 \times \mathbb{F}_2 \times \cdots \times \mathbb{F}_m \subseteq \mathbb{F}^m$ (i.e., its components come from gradually increasing extension fields). We refer to such points as *Matryoshka* points.[6]

We show that for a Matryoshka point r as above, it is possible to compute $\hat{f}(r)$ using a linear-size Boolean circuit. To see this, let us rework Eq. (1) as follows:

[6]Matryoshka dolls, popular in Russian culture, are a set of wooden dolls of increasing size placed one inside another, see https://en.wikipedia.org/wiki/Matryoshka_doll.

$$\begin{aligned}\hat{f}(r) &= \sum_{b\in\{0,1\}^m} eq(r,b)\cdot f(b)\\ &= \sum_{b\in\{0,1\}^m}\left(\prod_{i=1}^{m} eq_1(r_i,b_i)\right)\cdot f(b)\\ &= \sum_{b_m\in\{0,1\}} eq_1(r_m,b_m)\cdot \sum_{b_{m-1}\in\{0,1\}} eq_1(r_{m-1},b_{m-1})\cdots \sum_{b_1\in\{0,1\}} eq_1(r_1,b_1)\cdot f(b_1,\ldots,b_m), \qquad (2)\end{aligned}$$

where the equalities follow from from the definition of the multilinear extension and elementary algebraic manipulations.

Thus, we can compute $\hat{f}(r)$ using the following algorithm:

Matryoshka MLE Algorithm:

1. Initialize $f_0 = f$.
2. For $i = 1,\ldots,m$, compute $f_i : \{0,1\}^{m-i} \to \mathbb{F}_i$ as $f_i(b_{i+1},\ldots,b_m) = \sum_{b_i\in\{0,1\}} eq_1(r_i,b_i)\cdot f_{i-1}(b_i,\ldots,b_m)$ (this entails $O(2^{m-i})$ arithmetic operations over the field $\mathbb{F}_i$).
3. Output f_m (this is indeed a single scalar in $\mathbb{F}^m$).

The Matryoshka MLE algorithm precisely follows the RHS of Eq. (2), while computing the intermediate expressions from the innermost outward. Observe that iteration i only involves operations over the field $\mathbb{F}_i$ (since $r_i \in \mathbb{F}_i$ and the rest of the values are in $\mathbb{F}_{i-1}$). Thus, overall the algorithm can be implemented as a Boolean circuit of size:

$$\sum_{i=1}^{m} 2^{m-i}\cdot \mathrm{polylog}(|\mathbb{F}_i|),$$

and we have already set our fields to be such that this quantity is bounded by $O(2^m)$.

Remark 1.6. *Somewhat surprisingly, the Matryoshka MLE algorithm is actually also faster, by constant factors, than existing algorithms for computing a standard multilinear extension (i.e., when $r \in \mathbb{F}^m$).*

Specifically, we observe that Step 2 in the Matryoshka MLE algorithm can be implemented using exactly 2^{m-i} multiplications, since

$$\sum_{b_i\in\{0,1\}} eq_1(r_i,b_i)\cdot f_{i-1}(b_i,\ldots,b_m) = f_{i-1}(0,b_{i+1},\ldots,b_m) + r_i\cdot\big(f_{i-1}(1,b_{i+1},\ldots,b_m) - f_{i-1}(0,b_{i+1},\ldots,b_m)\big),$$

and so for each $b_{i+1},\ldots,b_m$, Step 2 requires only a single multiplication.

This leads to a total of $2^m - 1$ multiplications. The standard algorithm for this problem (see [Tha22, Lemma 3.8]) due to [VSBW13] requires $2^{m+1}-1$ multiplications, although we also mention that an algorithm implicit in [DT24] uses only $2^m + O(2^{m/2})$ multiplications.

Back to Proving MLE Claims. The Matyoshka MLE algorithm shows how to *compute* the MLE at a Matryoshka point, but what about proving its value? Recall that the prover is given as input $f : \{0,1\}^m \to \{0,1\}$ and a Matryoshka point $r \in \mathbb{F}_1 \times \cdots \times \mathbb{F}_m$. The verifier is given oracle access to an encoding, under a linear-time encodable code E of f, and the goal is to verify the value of $\hat{f}(r)$.

To do so we will choose $E = (E')^2$ to be a tensor product of a linear-time encodable code. Recall that the tensor code E is obtained from E' by viewing its messages as square matrices, and then encoding all of the rows via E' and the columns - both old and new - once again using E'.

Thus, the task at hand is to verify a particular linear combination of an encoded message. Following [RR24], if one arranges the coefficients of this linear combination as a square matrix, the matrix has rank 1 (basically because the multilinear extension is itself a tensor code). In other words, each coefficient is the product of two coefficients. One that is determined only by the row and the other only by the column. Thus, our approach will be to run the sumcheck protoocl for rank 1 coefficients.

The protocol for rank 1 coefficients from [RR24] (based on [Mei13]) relies on the sumcheck protocol, however it requires weighing the sums computed by the prover with the tensor coefficients which depend on the point r and come from large fields, and we do not generally know how to compute these weighted sums in linear time. To overcome this, we observe that in the special case that r has a Matryoshka structure, computing these weighted sums reduces to computing MLEs of certain functions at certain points that also have a Matryoshka structure, and so we can use the linear-time algorithm presented in the previous section to perform this task in linear time.

The above yields an interactive proof with $O(\sqrt{n})$ communication, but the idea can be extended to a d-dimensional tensor for any constant $d \geq 2$, reducing the communication complexity to $n^{1/d}$ and the verification complexity to $n^{O(1/d)}$. By composing the resulting interactive proof with a PCPP, one can further reduce the query complexity to $O(\log n)$ and the verification time to $\mathrm{polylog}(n)$.

To conclude, the logarithmic round interactive proof proceeds by first reducing the inner product to separate claims about the MLEs of x and y at Matryoshka points. Then these claims are established using a sumcheck for tensor codes, while exploiting the structure of the Matryoshka points.

Remark 1.7 (Comparison with [RR25])**.** *At a high-level, the protocol of [RR25] works by tensoring a large linear-time encodable tensor code with a tiny multiplication code. The protocol uses a sumcheck like step to reduce the size of the instance, and applies the code-switching technique of [RR24] to progress to the next round. In particular, the code-switching is applied in every step, to smaller and smaller instances.*

In contrast, our approach only involves a single application of code-switching (from a claim about the MLE to a claim about the tensor encoding). We find this protocol both simpler, and more efficient as it involves sending much shorter messages.

1.2.2 Inner product IOP with log star rounds

Our goal now is to reduce the number of rounds. The log-round protocol described above halves the instance size in each iteration, and therefore requires a logarithmic number of rounds in total. As an initial positive indication, observe that if we ran the procedure for only $O(\log m)$ many

rounds, the original instance, which has size 2^m, shrinks to size $\frac{2^m}{\text{poly}(m)}$. At this point we could solve the problem using an off-the-shelf tool such as the PCPP theorem, which has a poly-logarithmic overhead (via [BS08, Din07, Mie09]), which we can afford at this point.

This observation already reduces the number of rounds exponentially: from logarithmic to double logarithmic, but we show that we can do much better. Rather than making a sharp transition (i.e., initially reducing by half, and then abruptly reducing by the large remaining factor), we will attempt to make the transition smoother.

To do so, rather than considering the multilinear extension of functions, we will be looking at a different form of low-degree extension — one in which the individual degrees gradually increase. Thus, rather than considering Boolean functions $f, g : \{0,1\}^m \to \{0,1\}$, we consider functions $f, g : H_1 \times H_2 \times \cdots \times H_\ell \to \{0,1\}$, where each $H_i \subseteq \mathbb{F}_i$ is some subset of the corresponding field. Based on the previous discussion, we would like to have the H_i's be increasing in size so that we can have the number of variables ℓ (which also corresponds to the number of rounds) be much smaller than m (recall that m in the case of the warmup was logarithmic in the problem size).

In other words, we need to chose $H_1, \ldots, H_\ell$ such that $\prod_{i=1}^{\ell} H_i = 2^m$, and so that ℓ is as small as possible, and under some efficiency and soundness constraints about the H_i's that we need to analyze. We proceed directly to the analysis, and then chose the H_i's based on the induced constraints.

Let us now revisit the analog of the first step from the warmup: an inner product protocol in which the prover gets as input $f, g : H_1 \times \cdots \times H_\ell \to \{0,1\}$, the verifier gets as input the low degree extensions $\hat{f}, \hat{g} : \mathbb{F}_1 \times \cdots \times \mathbb{F}_\ell \to \mathbb{F}_\ell$, which are polynomials which agree with f and g on $H_1 \times \cdots \times H_\ell$, and have individual degree $|H_i| - 1$ in their i-th variable, for every $i \in [\ell]$. The goal of the verifier, as before, is to check that $\sum_{h \in H_1 \times \cdots \times H_\ell} f(x) \cdot g(x) = v$, for a given $v \in \{0,1\}$.

Following the sumcheck protocol, in each round i, the verifier now needs to generate the degree $2 \cdot (|H_i| - 1)$ polynomial:

$$p_i(\lambda) = \sum_{h \in H_{i+1} \times \cdots \times H_\ell} \hat{f}(r_1, \ldots, r_{i-1}, \lambda, h) \cdot \hat{g}(r_1, \ldots, r_{i-1}, \lambda, h),$$

where $r_1, \ldots, r_{i-1}$ is the randomness generated in the previous rounds.

To generate p_i, it suffices to compute its evaluation at $2|H_i| - 1$ values of λ. Let us assume that we have already pre-computed the two sequence of values $\left(\hat{f}(r_1, \ldots, r_{i-1}, h)\right)_{h \in H_i \times \cdots \times H_\ell}$ and $\left(\hat{g}(r_1, \ldots, r_{i-1}, h)\right)_{h \in H_i \times \cdots \times H_\ell}$ as in the usual linear-time sumcheck protocol.

Thus, to evaluate p_i at a point λ, we iterate over all $h \in H_{i+1} \times \cdots \times H_\ell$ and then compute the inner summand. Computing the inner summand requires interpolating two degree $|H_i| - 1$ univariate polynomials at a point $\lambda \in \mathbb{F}_i$, which can be done in $O(H_i)$ operations. Thus, the cost of computing $p_i(\lambda)$ is roughly $O(|H_i| \cdots |H_\ell|)$. Since we need to do so for $2|H_i| - 1$ values of λ, this leads to a total cost of roughly $O\left(|H_i|^2 \cdot |H_{i+1}| \ldots |H_\ell|\right)$, for round i.

Unfortunately, this actually does not seem very helpful toward our goal — in round i we manage to shrink the problem by a factor of $|H_i|$, but we cannot make $|H_{i+1}|$ much larger than $|H_i|$ since in the next round we will have a cost that is quadratic in it.

The last, simple but crucial, observation is that we can actually compute the round polynomials much faster. Evaluating $p_i(\lambda)$ separately for each value of λ is actually wasteful — it amounts to evaluating $\hat{f}(r_1, \ldots, r_{i-1}, \lambda, h)$ and $\hat{g}(r_1, \ldots, r_{i-1}, \lambda, h)$ at the $O(|H_i|)$ distinct points separately. But we know, via the FFT, that it is possible to evaluate a given univariate polynomial in time that is only *quasi-linear* in the number of evaluation points. Thus, we compute p_i as follows:

1. Fix a set Λ of $2|H_i| - 1$ evaluation points.
2. For $h \in H_{i+1} \times \cdots \times H_\ell$:
   (a) Observe that $\hat{f}(r_1, \ldots, r_{i-1}, \cdot, h)$ is a degree $|H_i| - 1$ univariate polynomial, and we know its evaluations on the points H_i. Thus, using the FFT, we generate the sequence $\left(\hat{f}(r_1, \ldots, r_{i-1}, \lambda, h)\right)_{\lambda \in \Lambda}$. We do the same for $\hat{g}$.[7] To do so, note that at the beginning of the round we have already computed the values $\left(\hat{f}(r_1, \ldots, r_{i-1}, h)\right)_{h \in H_i \times \cdots \times H_\ell}$ and $\left(\hat{g}(r_1, \ldots, r_{i-1}, h)\right)_{h \in H_i \times \cdots \times H_\ell}$.
   (b) For every $\lambda \in \Lambda$, update a running sum that computes $p_i(\lambda)$.

Notice that this algorithm is much faster — it runs in time $|H_i| \ldots |H_\ell| \cdot \text{polylog}(|H_i|)$ (we assume here that $|\mathbb{F}_i| \approx |H_i|$ so this accounts also for the cost of emulating the field operations).

This running time puts us at a much better position. Intuitively, the i-th round shaves off an $|H_i|$ factor from the input size, but will only have a $\text{polylog}(|H_{i+1}|)$ overhead in the next round. This suggests that we can afford for H_{i+1} to be *sub-exponentially* larger than H_i, while still ensuring that the cost of round $i+1$ is geometrically smaller than that of round i.

For sake of simplicity of the analysis, let us pretend that the running time for generating the i-th round polynomial is $|H_i| \ldots |H_\ell| \cdot \log(|H_i|)$ Boolean operations (i.e., the overhead is exactly logarithmic in $|H_i|$, rather than poly-logarithmic). In this case, we set H_1 to be a constant, and for every $i \in [\ell - 1]$ we set

$$|H_{i+1}| = 2^{|H_i|}/2.$$

Thus, the prover's computation time in round $i+1$ is:

$$|H_{i+1}| \cdots |H_\ell| \cdot \log(|H_{i+1}|) = |H_i| \cdots |H_\ell|/2,$$

which is less than half of the computation time in round i. Thus the total running time is a geometric series which converges to be linear in the cost of the initial round $O(|H_1| \cdots |H_\ell|) = O(2^m)$.

Let us turn to analyzing the number of rounds ℓ in the protocol. Observe that in the last round of the protocol we are shaving a factor that is roughly equivalent to ℓ repeated exponentiations (starting off from a constant). Since we cannot shave off more than the original input length, this shows that (up to the inaccuracies mentioned above) ℓ is at most the number of times that the log function can be applied to 2^m before it becomes a constant. That is, $O(\log^*(2^m))$.

Query reduction via PCPP compositon. The protocol, as described so far, requires the verifier to do work proportional to $|H_i|$ in the i-th round. In particular, this is the amount of queries that the verifier needs to do in each of the rounds. While this is fine for the first few rounds (when $|H_i|$ is small), for later rounds the cost becomes exorbitant (e.g., $|H_\ell|$ will be close to linear).

To reduce the number of queries, as well as the verification complexity, we compose the resulting interactive proof with a PCPP (see [RRR21, Section 7] or [ACY22]). This reduces the number of queries to be linear in the round complexity of the protocol. For this to to work, we need to make sure that none of the $|H_i|$'s is too large — in particular the last one. To ensure this, rather than

[7] To run the FFT we can either ensure that Λ is a suitable subgroup of the field, or alternatively, use general purpose results for multipoint evaluation, see Theorem 2.9. The former is significantly faster in practice, but as our focus is on simplicity, we prefer the latter option.

maintaining the exponential growth throughout the entire protocol, we cap them at a sufficient sub-linear size (say polylog(n)) and add a constant number of additional rounds to compensate for that.

This concludes the first step in the protocol — reducing the inner product claim to separate claims about low degree extensions at Matryoshka points. The second step — proving MLE evaluations at Matryoshka points — actually works almost exactly the same as in the case of the warmup (indeed notice that that protocol only had a constant number of rounds).

By combining the two steps we obtain an inner product protocol with $O(\log^*(n))$ rounds relative to a linear-time encodable code E.

Using the inner product protocol, the construction of an IOP for "regular" circuits follows using standard arithmetization techniques, see Section 5 for details.

1.3 Related works

As mentioned above, our works builds on prior IOPs for Boolean circuits [RR24, RR25, HR24]. We also note that Ron-Zewi and Weiss [RW24] extend the [RR24] protocol to be *zero-knowledge*. We leave open the question of whether their techniques can be combined with ours to construct a zero-knowledge IOP with a linear-time prover (and log star rounds).

Our use of higher degree extensions is similar, but also distinct, from other uses in the proof-systems literature. Notably, in their seminal work, Babai *et al.* [BFLS91] use a low degree extension in which the individual degree of each variable is poly-logarithmic, to make the corresponding low degree code have an inverse polynomial rate (whereas the multi-linear extension has an inverse *quasi*-polynomial rate). This setting of parameters has become typical in much of the PCP literature, but we emphasize that in these works all of the variables have the same degree. Somewhat closer to our technique is a technique used by [HR22], and later also by Gruen [Gru24], in which the first variable has a higher degree than the rest of the variables (in both works this is motivated by the fact that the original computation is expressed in a smaller base field).

Linear-time provers for arithmetic circuits over large finite fields were constructed in a sequence of works [BCG$^+$17, BCG20, GLS$^+$23, XZS22, BCF$^+$25, NST24, BMMS25]. Additionally, Bootle *et al.* [BCGL22] construct an IOP for computations over small fields, in which the prover is a random-access machine that runs in time linear in the circuit size. All of these works use (at least) a logarithmic number of rounds, prior to the application of Fiat-Shamir. We also mention the very recent works of Bünz *et al.* [BCFW25] and Baweja *et al.* [BMMS25] construct hash based accumulation schemes with linear-time provers.

1.4 Organization

Preliminaries are in Section 2. In Section 3 we give our new construction of an inner product IOP with a logarithmic number of rounds, and in Section 4 the one with log star rounds. In Section 5 we give a sketch of the (by now standard) construction of an IOP for Boolean circuits (or Boolean R1CS to be precise) from an inner product IOP.

2 Preliminaries

We will often view a string $w \in \Sigma^n$, over an alphabet Σ, as a function $w : [n] \to \Sigma$. In particular, the i-th entry of w is denoted $w(i)$. For strings $x, y \in \Sigma^n$, we let $\mathrm{dist}_\Sigma(x, y)$ denote the fraction of

coordinates $i \in [n]$ on which x and y differ, that is, $\mathrm{dist}_\Sigma(x,y) := |\{i \in [n] : x(i) \neq y(i)\}|/n$.

Following Diamond and Posen [DP24] we use Wiedemann's construction [Wie88] of a tower of binary fields. The fields in this construction are fields of characteristic 2, where the $(i+1)$-th field is the quadratic extension of the i-field. The important fact that we need about this construction is that there is a simple mapping of elements in the i-th field, to their representation in the $(i+1)$-th field.

The relevant needed facts are summarized in the following lemma.

Lemma 2.1 ([Wie88]). *There exists a sequence of finite fields $(\mathbb{F}_n)_{n\in\mathbb{N}}$, where $\mathbb{F}_i$ is the field of size 2^{2^n} such that:*

- *(Constructible:) Elements of $\mathbb{F}_n$ can be represented by $O(\log|\mathbb{F}_n|)$ bits, and given this representation, the field operations (addition, subtraction, multiplication, inversion and sampling random elements) can be done by $\mathrm{polylog}(|\mathbb{F}_n|)$ size Boolean circuits.*
- *(Extension:) There is a $\mathrm{polylog}(\mathbb{F}_n)$-size Boolean circuit that maps a representation of an element $\alpha \in \mathbb{F}_n$ to its representation as an element of the extension field $\mathbb{F}_{n+1}$.*

2.1 Interactive oracle proofs

We next define the notion of *interactive oracle proof*, due to [BCS16, RRR21]. We restrict our attention to the public-coin setting which means that all of the verifier's messages simply consist of uniformly random coins. Since we care about very small factors in the parties running times, the definition will be more detailed than usual. The definition is similar to the one given in [RR25, Section 2] (but slightly less detailed).

An ℓ-round (public-coin) interactive oracle protocol consists of two entities, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The prover $\mathcal{P}$ consists of ℓ Boolean circuits $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$. For every $i \in [\ell]$, the input to $\mathcal{P}_i$ is the state S_{i-1} from the previous round (where S_0 is simply the main input x and potentially also a witness w) as well as uniformly random coins R_{i-1}, which are generated by the verifier (where R_0 is defined as the empty string). The output of each circuit $\mathcal{P}_i$ is the state S_i for the next round and a message M_i to be transmitted to the verifier. The size $|\mathcal{P}|$ of the prover $\mathcal{P}$ is defined as the sum of the prover circuit sizes, i.e., $|\mathcal{P}| := |\mathcal{P}_1| + \cdots + |\mathcal{P}_\ell|$.

The verifier $\mathcal{V}$ is a Boolean circuit that given as input the transcript $(x, M_1, R_1, \ldots, M_{\ell-1}, R_{\ell-1}, M_\ell)$ decides whether to accept or reject. We will often be interested in verifiers that run in *sub-linear* time, and in particular are unable to read the entire transcript. To facilitate this, we consider pair languages for which the input is split into two parts $x = (x_{\mathrm{exp}}, x_{\mathrm{imp}})$. The first part, x_{exp} is read explicitly by the verifier (and will often consist of a parameterization of the problem). In contrast, the verifier only has *oracle access* to x_{imp}.[8] We model the verifier $\mathcal{V}$ as consisting of two separate circuits. The first circuit $\mathcal{V}_1$ takes as input $x_{\mathrm{exp}}, R_1, \ldots, R_{\ell-1}$, and outputs the set of query locations I. The circuit $\mathcal{V}_2$ then gets as input $x_{\mathrm{exp}}, R_1, \ldots, R_{\ell-1}$, as well as the projection of $(x_{\mathrm{imp}}, M_1, \ldots, M_\ell)$ to the query set I, denoted by $(x_{\mathrm{imp}}, M_1, \ldots, M_\ell)|_I$, and based on these decides whether to accept or reject. The size $|\mathcal{V}|$ of the verifier $\mathcal{V}$ is defined as the sum of the sizes of its constituent parts, i.e., $|\mathcal{V}| := |\mathcal{V}_1| + |\mathcal{V}_2|$.

The key parameters that we will care about are:

[8] The way the input is split is part of the specification of the language, and in all of our theorem statements we explicitly state how the input is split.

1. **Query Complexity:** the number of bits $q = |I|$ that the verifier reads from the input and transcript. An *interactive proof* corresponds to the case in which the verifier reads all of the transcript.
2. **Round complexity:** the number of rounds ℓ.
3. **Communication complexity:** The total length of P's messages $M_1, \ldots, M_\ell$.
4. **Randomness complexity:** The total length of V's messages $R_1, \ldots, R_\ell$.
5. **Verifier Size:** the size of the verifier $\mathcal{V}$, as defined above.
6. **Prover Size:** the size of the prover $\mathcal{P}$, as defined above. In the context of interactive oracle protocols for NP relations we will often assume that the prover is also given as an auxiliary input a witness w proving that the input x satisfies the relation.

Using the notion of interactive oracle protocols, we can now define interactive oracle proofs.

Definition 2.2 (Interactive oracle proof (IOP))**.** *An ℓ-round interactive oracle proof (IOP) with soundness error ε for a promise problem $(\mathsf{YES}, \mathsf{NO})$ is an ℓ-round (public-coin) interactive oracle protocol $(\mathcal{P}, \mathcal{V})$ such that:*

- **Completeness:** *If $x \in \mathsf{YES}$, then when $\mathcal{V}$ interacts with $\mathcal{P}$, it accepts with probability 1.*
- **Soundness:** *If $x \in \mathsf{NO}$, then for every prover strategy $\mathcal{P}^*$, when $\mathcal{V}$ interacts with $\mathcal{P}^*$, it accepts with probability at most ε over the verifier's random string $R_1, \ldots, R_{\ell-1}$.*

We call ε the soundness error of the IOP.

Focusing on *promise problems* allows us to model settings in which the input has some particular structure (e.g., is encoded under an error-correcting code). In particular, this will sometimes allow our verifier to run in time that is *sub-linear* even in the input.

Lastly, we note that the standard notion of PCP corresponds to the special case of IOP, when the round complexity is $\ell = 1$, while the notion of an *interactive proof* corresponds to the special case in which the verifier reads all of the transcript.

2.2 Error-correcting codes

Let Σ be a finite alphabet, and k, n be positive integers (the message length and the codeword length, respectively). An (error-correcting) code is an injective map $C : \Sigma^n \to \Sigma^{n'}$. The elements in the domain of C are called messages, and the elements in the image of C are called codewords. We say that C is systematic if the message is a prefix of the corresponding codeword, i.e., for every $x \in \Sigma^n$ there exists $z \in \Sigma^{n'-n}$ such that $C(x) = (x, z)$. If $\Sigma = \mathbb{F}^s$ for some finite field $\mathbb{F}$ and integer $s \geq 1$, and C is a linear map over $\mathbb{F}$, then we say that C is $\mathbb{F}$-linear. In the case that $\Sigma = \mathbb{F}$, we simply say that C is linear. The generating matrix of a linear code $C : \mathbb{F}^n \to \mathbb{F}^{n'}$ is a matrix $G \in \mathbb{F}^{n' \times n}$ such that $C(x) = G \cdot x$ for any $x \in \mathbb{F}^n$.

The rate of a code $C : \Sigma^n \to \Sigma^{n'}$ is the ratio $\rho := \frac{n}{n'}$. The relative distance $\mathrm{dist}(C)$ of C is the maximum $\delta > 0$ such that for every pair of distinct messages $x, y \in \Sigma^n$ it holds that $\mathrm{dist}_\Sigma(C(x), C(y)) \geq \delta$.

Below we mention some specific families of codes we will use in our protocols, and list their properties.

Tensor codes. A main ingredient in our constructions is the tensor product operation, defined as follows (see, e.g., [Sud01, DSW06]).

Definition 2.3 (Tensor codes)**.** *The* tensor product code *of linear codes $C_1 : \mathbb{F}^{n_1} \to \mathbb{F}^{n_1'}$ and $C_2 : \mathbb{F}^{n_2} \to \mathbb{F}^{n_2'}$ is the code $C \otimes C' : \mathbb{F}^{n_1 \times n_2} \to \mathbb{F}^{n_1' \times n_2'}$, where the encoding $(C_1 \otimes C_2)(M)$ of any message $M \in \mathbb{F}^{n_1 \times n_2}$ is obtained by first encoding each column of M with the code C_1, and then encoding each resulting row with the code C_2.*

Note that by linearity, the codewords of $C_1 \otimes C_2$ are $n_1 \times n_2$ matrices (over the field $\mathbb{F}$) whose columns belong to the code C_1, and whose rows belong to the code C_2. It is also known that the converse is true: any $n_1 \times n_2$ matrix, whose columns belong to the code C_1, and whose rows belong to the code C_2, is a codeword of $C_1 \otimes C_2$.

Fact 2.4. *A matrix $w \in \mathbb{F}^{n_1 \times n_2}$ is a codeword of $C_1 \otimes C_2$ if and only if the restriction of w to any column is a codeword of C_1, and the restriction of w to any row is a codeword of C_2.*

The following effects of the tensor product operation on the classical parameters of the code are well known.

Fact 2.5. *Suppose that $C_1 : \mathbb{F}^{n_1} \to \mathbb{F}^{n_1'}$, $C_2 : \mathbb{F}^{n_2} \to \mathbb{F}^{n_2'}$ are linear codes of rates ρ_1, ρ_2 and relative distances δ_1, δ_2 respectively. Then, the tensor product code $C_1 \otimes C_2$ is a linear code of rate $\rho_1 \cdot \rho_2$ and relative distance $\delta_1 \cdot \delta_2$. Moreover, if C_1, C_2 can be encoded by Boolean circuits of sizes s_1, s_2 respectively, then $C_1 \otimes C_2$ can be encoded by a Boolean circuit of size $n_1 \cdot s_1 + n_1 \cdot s_2$.*

For a linear code $C : \mathbb{F}^n \to \mathbb{F}^{n'}$, let $C^{\otimes 1} := C$ and $C^{\otimes t} := C \otimes C^{\otimes (t-1)}$, for any $t \geq 2$. As in the 2-dimensional case, the codewords of $C^{\otimes t} : \mathbb{F}^{n^t} \to \mathbb{F}^{(n')^t}$ can be viewed as t-dimensional cubes, satisfying that their projection on any axis-parallel line is a codeword of C. Once more, we have that the converse is also true.

Fact 2.6. *A t-dimensional cube $w \in \mathbb{F}^{(n')^t}$ is a codeword of $C^{\otimes t}$ if and only if the restriction of w to any axis-parallel line is a codeword of C.*

By applying Fact 2.5 iteratively, we get the following corollary.

Corollary 2.7. *If C is a linear code of rate ρ and relative distance δ, then $C^{\otimes t}$ is a linear code of rate ρ^t and relative distance δ^t. Furthermore, if C can be encoded by a Boolean circuit of size s, then $C^{\otimes t}$ can be encoded by a Boolean circuit of size $t(n')^{t-1}s$.*

Low-degree extension. One well-known example of tensor codes is obtained using the *low-degree extension*, defined as follows.

Fact 2.8 (Low-degree extension)**.** *Let $\mathbb{F}$ be a field and let $H_1, \ldots, H_m$ be arbitrary subsets of $\mathbb{F}$. Given a function $f : H_1 \times \ldots \times H_m \to \mathbb{F}$, there exists a unique m-variate polynomial $\hat{f} \in \mathbb{F}[x_1, \ldots, x_m]$ that agrees with f on $H_1 \times \cdots \times H_m$, and where each variable x_i has individual degree at most $|H_i| - 1$.*

We call $\hat{f}$ the low-degree extension of f. In the special case that $H_1 = \cdots = H_m = \{0, 1\}$, we call $\hat{f}$ the multilinear extension of f, and in the special case that $m = 1$, we call $\hat{f}$ the univariate extension of f.

We also use the following extension of the FFT, which allows to interpolate a given univariate polynomial in quasi-linear time, on any set of points.

Theorem 2.9 (Fast Evaluation of univariate extension (see [vzGG13, Chapter 10])). *Let $\mathbb{F}$ be a field and let $H \subseteq \mathbb{F}$ be an arbitrary subset. Given a function $f : H \to \mathbb{F}$, and points $\alpha_1, \ldots \alpha_{|H|} \in \mathbb{F}$ the values of the $(|H|-1)$-degree univariate extension $\hat{f}$ at the points $\alpha_1, \ldots, \alpha_{|H|}$ can be computed using $\widetilde{O}(|H|)$ field operations.*

Linear-time encodable codes. For our protocols, we shall also require linear-time encodable codes of distance arbitrarily close to 1, which can be obtained by combining the linear-time encodable codes of Spielman [Spi96] with the distance amplification method of [AEL95].

Theorem 2.10 (Linear-time encodable codes). *There exists an absolute constant σ_0 so that the following holds. For any $\varepsilon > 0$, and for any prime power $q \geq (1/\varepsilon)^{\sigma_0}$, there exists a family $\mathcal{C} = \{C_n : \mathbb{F}^n \to \mathbb{F}^{n'}\}_{n \in \mathbb{N}}$ of systematic linear codes of relative distance $1-\varepsilon$ over the field $\mathbb{F} = \mathbb{GF}(q)$ that can be encoded by a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$.*

Proof sketch. Theorem 3 of [GI05] shows how to combine the linear-time encodable codes of Spielman [Spi96] with the distance amplification method of [AEL95] to obtain $\mathbb{F}_q$-linear codes of relative distance $1 - \frac{\epsilon}{2}$ and alphabet size q^{Δ} for $q, \Delta = \mathrm{poly}(1/\epsilon)$, that can be encoded by a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$ (the structure of the alphabet and the dependence of the running time on ϵ are not explicitly stated in the theorem but can be deduced from the proof). By encoding each alphabet symbol in the resulting code with any explicit linear code over $\mathbb{F}_q$ of relative distance $1 - \frac{\epsilon}{2}$ (e.g., a Reed-Solomon code), one can obtain a linear code over $\mathbb{F}_q$ of relative distance at least $1 - \epsilon$ and encoding time $n \cdot \mathrm{poly}(1/\epsilon)$. □

3 Inner product check in log rounds

In this section, we prove the warmup version of our main result, as described in Section 1.2. Namely, a logarithmic-round *interactive proof* with a linear-size prover which reduces checking the inner product of a pair of strings to checking the value of an entry in the encoding of these strings via a linear-time encodable code C. Recall that a similar result was already shown in [RR25], but via a more complicated IOP protocol.

Theorem 3.1. *For any $\epsilon \in (0, \frac{1}{2})$ and constant $\gamma > 0$, there exists a field $\mathbb{F}_1$ of characteristic 2 and of size $\mathrm{poly}(1/\varepsilon)$, and a family $\mathcal{C} = \{C_n : (\mathbb{F}_1)^n \to (\mathbb{F}_1)^{n'}\}_{n \in \mathbb{N}}$ of linear codes of constant relative distance that can be encoded by a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$, so that the following holds.*

There exists an $O(\log n)$-round interactive protocol*, where the prover gets as input a pair of codewords $c_1 = C_n(x), c_2 = C_n(y) \in C_n$, where $x, y \in \mathbb{F}_1^n$, and a value $v \in \mathbb{F}_1$, and the verifier gets as input only the value v. At the end of the interaction, the verifier either rejects or outputs indices $i_1, i_2 \in [n']$ and values $u_1, u_2 \in \mathbb{F}_1$ such that:*

- ***Completeness:*** *If $\sum_{i \in [n]} x(i) \cdot y(i) = v$, then when V interacts with P it outputs $i_1, i_2 \in [n']$ and $u_1, u_2 \in \mathbb{F}_1$ such that $c_1(i_1) = u_1$ and $c_2(i_2) = u_2$.*
- ***Soundness:*** *If $\sum_{i \in [n]} x(i) \cdot y(i) \neq v$ then, for every P^*, when V interacts with P^*, with probability at least $1 - \varepsilon$, either V rejects or it outputs $i_1, i_2 \in [n']$ and $u_1, u_2 \in \mathbb{F}_1$ such that either $c_1(i_1) \neq u_1$ or $c_2(i_2) \neq u_2$.*

The protocol has communication complexity $n^{\gamma} \cdot \mathrm{poly}(1/\varepsilon)$ *and randomness complexity* $\log(n) \cdot O(\log\log(n) + 1/\varepsilon)$*. The prover can be implemented as a Boolean circuit of size* $n \cdot \mathrm{poly}(1/\varepsilon)$*, and the verifier has running time* $n^{O(\gamma)} \cdot \mathrm{poly}(1/\varepsilon)$*.*

Remark 3.2. *Theorem 3.1 can also be extended to checking the inner product of a constant number of strings instead of just two strings, however we state and prove it only for a pair of strings for simplicity of exposition.*

Also, using proof composition, the interactive proof given in the above theorem can be turned into an IOP *with logarithmic query complexity and polylogarithmic verifier running time, and the same communication complexity and prover size as in the above theorem (See Section 4.5, where we apply a similar transformation to our* $O(\log^* n)$*-round interactive proofs).*

Remark 3.3. *We remark that the polynomial dependence on* $1/\varepsilon$ *in the prover size in Theorem 3.1 can be improved by invoking the theorem wrt a constant* ε_0 *and applying (parallel) repetition or the more efficient amplification strategy from [HR22] to reduce the soundness error.*

Towards the proof of the above theorem, we first set in Section 3.1 below the parameters that will be used throughout this section. Then, in Section 3.2 we present a logarithmic-round interactive protocol with a linear-size prover which reduces checking the inner product of a pair of strings to checking the evaluation of their *multilinear extension* at a special point coming from a *Matryoshka series* (see Definition 3.4 in Section 3.1 below). In Section 3.3, we show a linear-time algorithm for computing the evaluation of the multilinear extension at Matryoshka points, and we use this algorithm in Section 3.4 to design a constant-round interactive protocol with a linear-size prover which reduces checking the evaluation of the multilinear extension at a Matryoshka point to checking the value of an entry in the encoding of the input strings via a linear-time encodable (tensor) code.

The above Theorem 3.1 then follows as a direct corollary of Lemma 3.5 from Section 3.2 and Lemma 3.10 from Section 3.4.

3.1 Setting of parameters

Our goal is to design a protocol that given a pair of strings $x, y \in \mathbb{F}_1^n$ and a value $v \in \mathbb{F}_1$, checks that $\sum_{i\in[n]} x(i) \cdot y(i) = v$. In what follows, it will be convenient for us to view the strings $x, y \in \mathbb{F}_1^n$ as (the truth table of) functions $f, g : \{0,1\}^m \to \mathbb{F}_1$, respectively, for $m = \lceil \log n \rceil$. Under this notation, our goal is to check that $\langle f, g\rangle := \sum_{b\in\{0,1\}^m} f(b) \cdot g(b) = v$.

Our protocols use a special kind of increasing field ensemble, that we call *Matryoshka series*.

Definition 3.4 (Matryoshka series)**.** *Let* $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$ *be an ensemble of finite fields. We say that* $\mathbb{F}$ *is* increasing *if* $\mathbb{F}_t \subseteq \mathbb{F}_{t+1}$ *for every* $t \in \mathbb{N}$*. Let* $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$ *be an increasing fields ensemble. Then its* Matryoshka series *is the sequence* $\mathcal{M}^{\mathbb{F}} = (\mathcal{M}_t^{\mathbb{F}})_{t\in\mathbb{N}}$*, where* $\mathcal{M}_t^{\mathbb{F}} = \mathbb{F}_1 \times \mathbb{F}_2 \times \cdots \times \mathbb{F}_t$*.*

When the field ensemble $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$ is clear from the context, we shall omit it from the notation of $\mathcal{M}^{\mathbb{F}}$. We emphasize that the notation $\mathbb{F}_t$ simply refers to the t-th field in the ensemble, whereas $\mathrm{GF}(a)$ refers to the finite field containing a elements.

We shall use a specific choice of field ensemble $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$, defined as follows. Let σ_0 be a sufficiently large constant to be determined later on, let $\varepsilon \in (0, \frac{1}{2})$ be a parameter, and let $\gamma > 0$ be a constant. For $t \in \mathbb{N}$, let a_t be the smallest integer of the form 2^{2^j} that is larger than $\frac{t^2}{\varepsilon^{\sigma_0}}$, and let $\mathbb{F}_t = \mathrm{GF}(a_t)$ be the field of a_t elements. Note that for every $t \in \mathbb{N}$, it holds that

$$\frac{t^2}{(\varepsilon\gamma)^{\sigma_0}} < a_t \leq \left(\frac{t^2}{(\varepsilon\gamma)^{\sigma_0}}\right)^2. \tag{3}$$

Also note that $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$ is an increasing field ensemble, where $\mathbb{F}_t$ is a field of characteristic 2 and size a_t. We let $\mathcal{M} := \mathcal{M}^{\mathbb{F}} = (\mathcal{M}_t)_{t\in\mathbb{N}}$ denote its Matryoshka series.

3.2 Reducing Inner Product Claim to Matryoshka points in MLE

In this section, we present a logarithmic-round interactive protocol with a linear-size prover which reduces checking the inner product of a pair of functions $f, g : \{0,1\}^m \to \mathbb{F}_1$ to checking the evaluation of their multilinear extensions at a special point coming from the Matryoshka series defined in the previous section.

Lemma 3.5. *Let $\epsilon \in (0, \frac{1}{2})$ be a parameter, let $\gamma > 0$ be a constant, and let $\mathbb{F}_t$ and $\mathcal{M}_t$ be defined as in Section 3.1 with respect to ϵ and γ. Then there exists an $(m+1)$-round interactive protocol, where the prover gets as input a pair of functions $f, g : \{0,1\}^m \to \mathbb{F}_1$ and a value $v \in \mathbb{F}_1$, and the verifier gets as input only the value v. At the end of the interaction, the verifier either rejects or outputs a point $r \in \mathcal{M}_m$ and values $\alpha, \beta \in \mathbb{F}_m$ such that:*

- ***Completeness:*** *If $\langle f, g\rangle = v$, then when V interacts with P it outputs $r \in \mathcal{M}_m$ and $\alpha, \beta \in \mathbb{F}_m$ such that $\hat{f}(r) = \alpha$ and $\hat{g}(r) = \beta$, where $\hat{f}, \hat{g} : (\mathbb{F}_m)^m \to \mathbb{F}_m$ denote the multilinear extension of f, g, respectively, when viewed as functions $f, g : \{0,1\}^m \to \mathbb{F}_m$.*
- ***Soundness:*** *If $\langle f, g\rangle \neq v$ then, for every P^*, when V interacts with P^*, with probability at least $1 - \varepsilon$, either V rejects or it outputs $r \in \mathcal{M}_m$ and $\alpha, \beta \in \mathbb{F}_m$ such that $\hat{f}(r) \neq \alpha$ or $\hat{g}(r) \neq \beta$.*

The protocol has communication and randomness complexity at most $m \cdot O(\log(m/\varepsilon))$, the prover can be implemented as a Boolean circuit of size $2^m \cdot \mathrm{polylog}(1/\varepsilon)$, and the verifier has running time $m \cdot \mathrm{polylog}(m/\varepsilon)$.

Proof. The protocol establishing Lemma 3.5 follows along the lines of the classical sumcheck protocol [LFKN92], except that it uses increasingly large fields from our carefully chosen field ensemble $\mathbb{F}$. The formal description of the protocol is given in Fig. 1. We proceed to show that it satisfies the requirements.

Completeness. We start by proving the following claim:

Claim 3.6. *For every $t \in \{0, 1, \ldots, m\}$ and $b_{t+1}, \ldots, b_m \in \{0,1\}$ it holds that*

$$f_t(b_{t+1}, \ldots, b_m) = \hat{f}(r_1, \ldots, r_t, b_{t+1}, \ldots, b_m)$$

and

$$g_t(b_{t+1}, \ldots, b_m) = \hat{g}(r_1, \ldots, r_t, b_{t+1}, \ldots, b_m).$$

Proof. We prove the claim for f, the proof for g is identical.

The proof is by induction over t. For $t = 0$, this is true by our setting of $f_0 = f$. Assuming the claim holds for $t-1$, we prove it holds for t. By definition of f_t in Step 3e, we have that

[8] We assume without loss of generality that n is a power of 2, otherwise padding with zeros at most doubles the input size.

Prover Input: A pair of functions $f, g : \{0,1\}^m \to \mathbb{F}_1$, a value $v \in \mathbb{F}_1$.
Verifier Input: The value $v \in \mathbb{F}_1$.

The Protocol:

1. Let $\Lambda \subseteq \mathbb{F}_1$ be an arbitrary subset of three distinct field elements such that $0, 1 \in \Lambda$.
2. Set $v_0 := v$, $f_0 =: f$, $g_0 := g$.
3. For $t = 1, 2, \ldots, m$:
   ▷ **Invariant**: *At the beginning of iteration t, the functions $f_{t-1}, g_{t-1} : \{0,1\}^{m-(t-1)} \to \mathbb{F}_{t-1}$ and scalar $v_{t-1} \in \mathbb{F}_{t-1}$ have already been computed (where we set $\mathbb{F}_0 := \mathbb{F}_1$). Recalling that $\mathbb{F}_{t-1}$ is a subfield of $\mathbb{F}_t$, in what follows, we view the range of f_{t-1} and g_{t-1} as being $\mathbb{F}_t$.*
   (a) The prover P computes and sends to V the function $w_t : \Lambda \to \mathbb{F}_t$ defined as
   $$w_t(\lambda) = \sum_{b_{t+1},\ldots,b_m \in \{0,1\}} \hat{f}_{t-1}(\lambda, b_{t+1}, \ldots, b_m) \cdot \hat{g}_{t-1}(\lambda, b_{t+1}, \ldots, b_m),$$
   for every $\lambda \in \Lambda$, where $\hat{f}_{t-1}, \hat{g}_{t-1} : (\mathbb{F}_t)^{m-(t-1)} \to \mathbb{F}_t$ denote the multilinear extensions of $f_{t-1}, g_{t-1} : \{0,1\}^{m-(t-1)} \to \mathbb{F}_t$, respectively.
   (b) V checks that $w_t(0) + w_t(1) = v_{t-1}$, otherwise it rejects.
   (c) V randomly chooses $r_t \in \mathbb{F}_t$ and sends it to P.
   (d) V computes $v_t = \hat{w}_t(r_t)$, where $\hat{w}_t : \mathbb{F}_t \to \mathbb{F}_t$ denotes the univariate extension of $w_t : \Lambda \to \mathbb{F}_t$ (a degree 2 polynomial).
   (e) P computes $f_t, g_t : \{0,1\}^{m-t} \to \mathbb{F}_t$, defined as $f_t(b_{t+1}, \ldots, b_m) = \hat{f}_{t-1}(r_t, b_{t+1}, \ldots, b_m)$ and $g_t(b_{t+1}, \ldots, b_m) = \hat{g}_{t-1}(r_t, b_{t+1}, \ldots, b_m)$ for all $b_{t+1}, \ldots, b_m \in \{0,1\}$.
4. P sends $\alpha := f_m \in \mathbb{F}_m$ and $\beta := g_m \in \mathbb{F}_m$.
5. V checks that $\alpha \cdot \beta = v_m$, otherwise it rejects.
6. V outputs $r = (r_1, \ldots, r_m) \in \mathcal{M}_m$ and $\alpha, \beta \in \mathbb{F}_m$.

Figure 1: Matroyshka Inner Product Check

$f_t(b_{t+1},\ldots,b_m) = \hat{f}_{t-1}(r_t, b_{t+1},\ldots,b_m)$ for any $b_{t+1},\ldots,b_m \in \{0,1\}$. On the other hand, by the induction hypothesis we have that for any $b_t,\ldots,b_m \in \{0,1\}$,

$$\hat{f}_{t-1}(b_t,\ldots,b_m) = f_{t-1}(b_t,\ldots,b_m) = \hat{f}(r_1,\ldots,r_{t-1},b_t,\ldots,b_m).$$

So $\hat{f}_{t-1}(x_t,\ldots,x_m)$ and $\hat{f}(r_1,\ldots,r_{t-1},x_t,\ldots,x_m)$ are two multilinear polynomials over $\mathbb{F}_m$ in $m-(t-1)$ indeterminates $x_t,\ldots,x_m$ that agree on all values in $\{0,1\}^{m-(t-1)}$, and so these polynomials must be identical. We conclude that for any $b_{t+1},\ldots,b_m \in \{0,1\}$,

$$f_t(b_{t+1},\ldots,b_m) = \hat{f}_{t-1}(r_t, b_{t+1},\ldots,b_m) = \hat{f}(r_1,\ldots,r_t,b_{t+1},\ldots,b_m).$$

□

Completeness relies on the following claim.

Claim 3.7. *For any $t \in \{1,2,\ldots,m-1\}$, it holds that $w_{t+1}(0) + w_{t+1}(1) = \hat{w}_t(r_t)$.*

Proof. Fix $t \in \{1,2,\ldots,m-1\}$. Then we have that

$$\begin{aligned}
w_{t+1}(0) + w_{t+1}(1) &= \sum_{b\in\{0,1\}} \sum_{b_{t+2},\ldots,b_m\in\{0,1\}} \hat{f}_t(b, b_{t+2}\ldots,b_m)\cdot \hat{g}_t(b, b_{t+2},\ldots,b_m) \\
&= \sum_{b_{t+1},\ldots,b_m\in\{0,1\}} f_t(b_{t+1},\ldots,b_m)\cdot g_t(b_{t+1},\ldots,b_m) \\
&= \sum_{b_{t+1},\ldots,b_m\in\{0,1\}} \hat{f}_{t-1}(r_t, b_{t+1},\ldots,b_m)\cdot \hat{g}_{t-1}(r_t, b_{t+1},\ldots,b_m) \\
&= \widehat{w}_t(r_t),
\end{aligned}$$

where the last equality follows since $\widehat{w}_t(x)$ and

$$\sum_{b_{t+1},\ldots,b_m\in\{0,1\}} \hat{f}_{t-1}(x, b_{t+1},\ldots,b_m)\cdot \hat{g}_{t-1}(x, b_{t+1},\ldots,b_m)$$

are both degree 2 polynomials in the indeterminate x, which by Step 3a, agree on all three points $\lambda \in \Lambda$, hence they must be the same polynomial. □

Next assume that $\langle f, g\rangle = v$. Then we have that

$$\begin{aligned}
w_1(0) + w_1(1) = &\sum_{b\in\{0,1\}} \sum_{b_2,\ldots,b_m\in\{0,1\}} \hat{f}_0(b, b_2\ldots,b_m)\cdot \hat{g}_0(b, b_2,\ldots,b_m) \\
&\sum_{b_1,b_2,\ldots,b_m\in\{0,1\}} f_0(b_1, b_2\ldots,b_m)\cdot g_0(b_1, b_2,\ldots,b_m) = v_0,
\end{aligned}$$

and so the verifier does not reject on Step 3b of the first iteration. Moreover, by Claim 3.7 above, the verifier also does not reject on Step 3b in any of the iterations $t \in \{2,\ldots,m\}$. Furthermore, we have that

$$\alpha\cdot\beta = f_m\cdot g_m = \hat{f}_{m-1}(r_m)\cdot\hat{g}_{m-1}(r_m) = \widehat{w}_m(r_m) = v_m,$$

where the third equality follows by the same argument as in the proof of Claim 3.7, and so the verifier does not reject on Step 5 as well. Finally, by Claim 3.6, the verifier outputs α, β, and r which satisfy that $\alpha = f_m = \hat{f}(r)$ and $\beta = g_m = \hat{g}(r)$.

Soundness. Next assume that $\langle f, g\rangle \neq v$. Fix a prover strategy P^*, and denote by $w_1^*, w_2^*, \ldots, w_m^*$ the messages that P^* sends in Step 3a. Soundness relies on the following claim.

Claim 3.8. *If for some $t \in \{1, 2, \ldots, m-1\}$, it holds that $\hat{w}_t^*(r_t) \neq \hat{w}_t(r_t)$, and V does not reject in Step 3b in iteration $t+1$, then with probability at least $1 - \frac{2}{|\mathbb{F}_{t+1}|}$ over the choice of r_{t+1}, it holds that $\hat{w}_{t+1}^*(r_{t+1}) \neq \hat{w}_{t+1}(r_{t+1})$.*

Proof. Fix $t \in \{1, \ldots, m-1\}$, and assume that $\hat{w}_t^*(r_t) \neq \hat{w}_t(r_t)$, and that V does not reject in Step 3b in iteration $t+1$. By assumption that V does not reject in Step 3b of iteration $t+1$, we have that $w_{t+1}^*(0) + w_{t+1}^*(1) = \widehat{w}_t^*(r_t)$. On the other hand, by Claim 3.7 we know that $w_{t+1}(0) + w_{t+1}(1) = \widehat{w}_t(r_t)$. Therefore, by assumption that $\hat{w}_t^*(r_t) \neq \hat{w}_t(r_t)$, we have that $w_{t+1}^*(0) + w_{t+1}^*(1) \neq w_{t+1}(0) + w_{t+1}(1)$. Hence $\widehat{w}_{t+1}$ and $\widehat{w}_{t+1}^*$ are two distinct degree 2 polynomials over $\mathbb{F}_{t+1}$, and so they agree on at most a $\frac{2}{|\mathbb{F}_{t+1}|}$-fraction of the points. Thus with probability at least $1 - \frac{2}{|\mathbb{F}_{t+1}|}$ over the choice of r_{t+1}, it holds that $\widehat{w}_{t+1}(r_{t+1}) \neq \widehat{w}_{t+1}^*(r_{t+1})$. □

Now, if the verifier rejects in any of the iterations then we are done. Hence we may assume that the verifier does not reject in any of the iterations.

By assumption that $\langle f, g\rangle \neq v$, we have that

$$w_1(0) + w_1(1) = \sum_{b_1, b_2, \ldots, b_m \in \{0,1\}} f_0(b_1, b_2 \ldots, b_m) \cdot g_0(b_1, b_2, \ldots, b_m) \neq v_0.$$

On the other hand, by assumption that the verifier does not reject on Step 3b, we also have that $w_1^*(0) + w_1^*(1) = v_0$. We conclude that $\hat{w}_1$ and $\hat{w}_1^*$ are two distinct degree 2 polynomials, and by the same argument as in the proof of Claim 3.8, this implies in turn that $\hat{w}_1(r_1) \neq \hat{w}_1^*(r_1)$ with probability at least $1 - \frac{2}{|\mathbb{F}_1|}$ over the choice of r_1. Applying Claim 3.8 above iteratively, and using the union bound, we then conclude that with probability at least $1 - \sum_{t=1}^{m} \frac{2}{|\mathbb{F}_t|}$, it holds that $\hat{w}_m(r_m) \neq \hat{w}_m^*(r_m)$.

Next assume that this latter event holds, i.e., that $\hat{w}_m(r_m) \neq \hat{w}_m^*(r_m)$, and let α^*, β^* denote the messages sent by P on Step 4. Then assuming that the verifier does not reject on Step 5, we have that $\alpha^* \cdot \beta^* = \hat{w}_m^*(r_m)$. On the other hand, by Claim 3.6 we have that $\hat{w}_m(r_m) = f_m \cdot g_m = \hat{f}(r) \cdot \hat{g}(r)$. We conclude that in this case $\alpha^* \cdot \beta^* \neq \hat{f}(r) \cdot \hat{g}(r)$, and so either $\hat{f}(r) \neq \alpha^*$ or $\hat{g}(r) \neq \beta^*$.

Overall, we obtain a soundness error of at most

$$\sum_{t=1}^{m} \frac{2}{|\mathbb{F}_t|} \leq \sum_{t=1}^{\infty} \frac{2}{|\mathbb{F}_t|} = \sum_{t=1}^{\infty} \frac{2}{a_t} \leq \sum_{t=1}^{\infty} \frac{2(\varepsilon\gamma)^{\sigma_0}}{t^2} \leq \varepsilon^{\sigma_0} \cdot \sum_{t=1}^{\infty} \frac{2}{t^2} < \varepsilon,$$

where the last inequality follows for a sufficiently large constant σ_0.

Number of rounds is clearly $m+1$. Next we analyze the communication and randomness complexity, and prover and verifier running time.

Communication and Randomness Complexity. In every round t, both the prover and the verifier send a constant number of elements of $\mathbb{F}_t$, where each field element can be represented using $O(\log(|\mathbb{F}_t|))$ bits. Hence there exists an absolute constant ξ_0 so that the total communication and randomness complexity is at most

$$\sum_{t=1}^{m} \xi_0 \cdot \log(|\mathbb{F}_t|) = \sum_{t=1}^{m} \xi_0 \cdot \log(a_t) \leq m \cdot O(\log(a_m)) \leq m \cdot O(\log(m/\epsilon)).$$

Prover complexity. In every round t, in both Steps 3a and 3e, the prover has to compute the values of $\hat{f}_{t-1}$ and $\hat{g}_{t-1}$ on 2^{m-t} points of the form $(\lambda, b_{t+1}, \ldots, b_m)$, where $\lambda \in \mathbb{F}_t$ and $b_{t+1}, \ldots, b_m \in \{0,1\}$. We claim that each such value can be computed using a constant number of field operations over $\mathbb{F}_t$. To see this, note that for any $b_{t+1}, \ldots, b_m \in \{0,1\}^m$ we have that $\hat{f}_{t-1}(x, b_{t+1}, \ldots, b_m)$ is a degree 1 polynomial over $\mathbb{F}_t$ in the indeterminate x, and hence its value on any point in $\mathbb{F}_t$ can be reconstructed from $f_{t-1}(0, b_{t+1}, \ldots, b_m)$ and $f_{t-1}(1, b_{t+1}, \ldots, b_m)$ using a constant number of field operations in $\mathbb{F}_t$ (and similarly for $\hat{g}_{t-1}$). Hence, all the required values can be computed using $O(2^{m-t})$ field operations in $\mathbb{F}_t$. Additionally, the prover has to use $O(2^{m-t})$ field operations in $\mathbb{F}_t$ to compute the inner product on Step 3a from these values.

Since each field operation in $\mathbb{F}_t$ can be performed in time $\text{polylog}(|\mathbb{F}_t|)$, overall there exists an absolute constant ξ_0 so that the prover can be implemented as a Boolean circuit of size:

$$\begin{aligned}
\sum_{t=1}^{m} 2^{m-t} \cdot log^{\xi_0}(|\mathbb{F}_t|) &= 2^m \sum_{t=1}^{m} \frac{\log^{\xi_0}(a_t)}{2^t} \\
&\leq 2^m \sum_{t=1}^{\infty} \frac{\log^{2\xi_0}(t/\epsilon)}{2^t} \\
&\leq 2^m \sum_{t=1}^{\infty} \frac{\log^{2\xi_0}(t) + \log^{2\xi_0}(1/\varepsilon)}{2^t} \\
&\leq 2^m \cdot \text{polylog}(1/\varepsilon).
\end{aligned}$$

Verifier complexity. In every round t, Step 3b can be clearly performed using a constant number of field operations in $\mathbb{F}_t$. In Step 3d, the verifier needs to compute the value of $\widehat{w}_t$ on $r_t \in \mathbb{F}_t$. Since $\widehat{w}_t$ is a degree 2 polynomial, this value can be computed from the values $w_t(\lambda)$ for $\lambda \in \Lambda$ using a constant number of field operations in $\mathbb{F}_t$. So overall, the verifier performs a constant number of field operations in $\mathbb{F}_t$.

Therefore, there exists an absolute constant ξ_0 so that the overall verifier running time is at most

$$\sum_{t=1}^{m} \log^{\xi_0}(|\mathbb{F}_t|) = \sum_{t=1}^{m} \log^{\xi_0}(a_t) \leq m \cdot \text{polylog}(a_m) \leq m \cdot \text{polylog}(m/\epsilon).$$

□

3.3 Computing a Matryoshka point in the MLE

Next we show that the multilinear extension of a given function $f : \{0,1\}^m \to \mathbb{F}_1$ can be evaluated in *linear time* on points coming from the Matryoshka series corresponding to our choice of field ensemble.

Lemma 3.9. *Let $\epsilon \in (0, \frac{1}{2})$ be a parameter, let $\gamma > 0$ be a constant, and let $\mathbb{F}_t$ and $\mathcal{M}_t$ be defined as in Section 3.1 with respect to ϵ and γ. Then there exists a Boolean circuit of size $2^m \cdot \text{polylog}(1/\varepsilon)$ that given as input a function $f : \{0,1\}^m \to \mathbb{F}_1$ and a point $r \in \mathcal{M}_m$, outputs $\hat{f}(r) \in \mathbb{F}_m$, where $\hat{f}$ denotes the multilinear extension of f, when viewed as a function $f : \{0,1\}^m \to \mathbb{F}_m$.*

Proof. The algorithm is presented in Fig. 2. Note that precisely the same procedure as in Figure 2 was used in the protocol of Figure 1 by the prover to compute the value of $\hat{f}(r)$, and exactly

Input: A function $f : \{0,1\}^m \to \mathbb{F}_1$ and a point $r = (r_1, \ldots, r_m) \in \mathcal{M}_m$.
Output: $\hat{f}(r)$.

The Algorithm:

1. Set $f_0 := f$.
2. For $t = 1, 2, \ldots, m$:
   ▷ **Invariant**: *At the beginning of iteration t, the function $f_{t-1} : \{0,1\}^{m-(t-1)} \to \mathbb{F}_{t-1}$ has already been computed (where we set $\mathbb{F}_0 := \mathbb{F}_1$). Recalling that $\mathbb{F}_{t-1}$ is a subfield of $\mathbb{F}_t$, in what follows, we view the range of f_{t-1} as being $\mathbb{F}_t$.*

   Compute $f_t : \{0,1\}^{m-t} \to \mathbb{F}_t$, defined as $f_t(b_{t+1}, \ldots, b_m) = \hat{f}_{t-1}(r_t, b_{t+1}, \ldots, b_m)$ for all $b_{t+1}, \ldots, b_m \in \{0,1\}$, where $\hat{f}_{t-1} : (\mathbb{F}_t)^{m-(t-1)} \to \mathbb{F}_t$ denotes the multilinear extension of $f_{t-1} : \{0,1\}^{m-(t-1)} \to \mathbb{F}_t$.
3. Output $f_m \in \mathbb{F}_m$.

Figure 2: Fast MLE Evaluation at Matroyshka Points

the same analysis shows that $f_m = \hat{f}(r)$, and that the algorithm can be implemented as a Boolean circuit of size $2^m \cdot \text{polylog}(1/\varepsilon)$.

□

3.4 Fast code switching for MLE

In this section, we use the algorithm from the previous section to design a constant-round interactive protocol with a linear-size prover which reduces checking the evaluation of the multilinear extension of a given function $f : \{0,1\}^m \to \mathbb{F}_1$ at a Matryoshka point to checking the value of an entry in the encoding of (the truth table of) f via a linear-time encodable code.

Lemma 3.10. *Let $\epsilon \in (0, \frac{1}{2})$ be a parameter, let $\gamma > 0$ be a constant, and let $\mathbb{F}_t$ and $\mathcal{M}_t$ be defined as in Section 3.1 with respect to ϵ and γ. Then there exists a systematic linear code ensemble $\mathcal{C} = \{(\mathbb{F}_1)^n \to (\mathbb{F}_1)^{n'}\}_{n \in \mathbb{N}}$ that can be encoded using a Boolean circuit of size $n \cdot \text{poly}(1/\epsilon)$, and a constant-round interactive protocol with the following properties:*

- ***Prover's input:*** *The prover gets as input a point $r \in \mathcal{M}_m$, a value $v \in \mathbb{F}_m$, and a codeword $c = \mathcal{C}(f)$ for some function $f : \{0,1\}^m \to \mathbb{F}_1$, where f is viewed as a length 2^m binary string.*
- ***Verifier's input:*** *The verifier gets as input the point $r \in \mathcal{M}_m$ and the value $v \in \mathbb{F}_m$.*
- ***Completeness:*** *If $\hat{f}(r) = v$, then when V interacts with P, it outputs an entry ρ and a value $u \in \mathbb{F}_1$ so that $c(\rho) = u$, where $\hat{f}$ denotes the multilinear extension of f, when viewed as a function $f : \{0,1\}^m \to \mathbb{F}_m$.*
- ***Soundness:*** *If $\hat{f}(r) \neq v$ then, for every P^*, when V interacts with P^*, with probability at least $1 - \varepsilon$, it either rejects or outputs ρ and $u \in \mathbb{F}_1$ so that $c(\rho) \neq u$.*

The protocol has communication complexity $2^{\gamma m} \cdot \mathrm{poly}(1/\varepsilon)$ *and randomness complexity* $m + O(\log(1/\varepsilon))$*. The prover can be implemented as a Boolean circuit of size* $2^m \cdot \mathrm{poly}(1/\varepsilon)$*, and the verifier has running time* $2^{O(\gamma m)} \cdot \mathrm{poly}(1/\varepsilon)$*.*

Proof. The protocol establishing Lemma 3.10 relies on the code switching technique of [RR24], and uses the algorithm for fast MLE evaluation at Matryoshka points from the previous section to compute the weighted sums in the protocol of [RR24] in linear time.

The code ensemble $\mathcal{C}$: Let $\ell := \lceil 2/\gamma \rceil$. Let $\{C_n : (\mathbb{F}_1)^n \to (\mathbb{F}_1)^{n'}\}_{n\in\mathbb{N}}$ be the systematic linear code ensemble of relative distance $1 - \frac{\epsilon}{\ell}$ that is encodable using a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$, guaranteed by Theorem 2.10 for a sufficiently large constant σ_0, and let $\mathcal{C} = \{(C_n)^{\otimes \ell}\}_{n\in\mathbb{N}}$. Note that by Fact 2.6, $(C_n)^{\otimes \ell} : (\mathbb{F}_1)^{n^\ell} \to (\mathbb{F}_1)^{(n')^\ell}$ can be encoded in time $n^\ell \cdot \mathrm{poly}(1/\varepsilon)$.

The encoding of a function $f : \{0,1\}^m \to \mathbb{F}_1$ via $\mathcal{C}$ is performed as follows. Without loss of generality, we may assume that ℓ divides m (otherwise, we can increase the number of variables by at most ℓ, and embed the function f on all inputs that assign zeros to these additional variables), and let $n := \frac{m}{\ell}$. Let $N := 2^n$, and for $i \in [N]$, let $\bar{i} \in \{0,1\}^n$ denote its binary representation. We view f as a string $M_f \in (\mathbb{F}_1)^{N^\ell}$ (noting that $N^\ell = 2^{n\cdot\ell} = 2^m$) which satisfies that $M_f(i_1, \ldots, i_\ell) = f\left(\overline{i_1}, \ldots, \overline{i_\ell}\right)$ for any $i_1, \ldots, i_\ell \in [N]$. The encoding $\mathcal{C}(f)$ of f is $(C_N)^{\otimes \ell}(M_f) \in (\mathbb{F}_1)^{(N')^\ell}$.

The interactive protocol: The formal description of the protocol is presented in Fig. 3. In what follows, every t, we view $\mathbb{F}_t$ as a $\log_{a_1}(a_t)$-dimensional vector space over $\mathbb{F}_1$ (recalling that $\mathbb{F}_1 \subseteq \mathbb{F}_t$), and fix a basis B_t of the field $\mathbb{F}_t$ over $\mathbb{F}_1$. Every element $\alpha \in \mathbb{F}_t$ can be uniquely expressed as a linear combination $\alpha = \sum_{b\in B_t} \alpha|_b \cdot b$ where $\alpha|_b \in \mathbb{F}_1$. More generally, any vector $w \in (\mathbb{F}_t)^n$ can be uniquely expressed as a linear combination $w = \sum_{b\in B_t} w|_b \cdot b$, where $w|_b \in (\mathbb{F}_1)^n$.

We proceed to show that the protocol satisfies the requirements.

Completeness. Completeness relies on the following claim.

Claim 3.11. *It holds that*

$$\hat{y}_1(r_{n\cdot(\ell-1)+1}, \ldots, r_{n\cdot\ell}) = \hat{f}(r_1, \ldots, r_{n\cdot\ell}),$$

and for any $t \in \{1, 2, \ldots, \ell - 1\}$*,*

$$\hat{y}_{t+1}(r_{n\cdot(\ell-(t+1))+1}, \ldots, r_{n\cdot(\ell-t)}) = w_t(\rho_{\ell-(t-1)}).$$

Proof. We first show that $\hat{y}_1(r_{n\cdot(\ell-1)+1}, \ldots, r_{n\cdot\ell}) = \hat{f}(r_1, \ldots, r_{n\cdot\ell})$. For every $i_1, \ldots, i_\ell \in [N]$, it holds that

$$f_{1,i_\ell}\left(\overline{i_1}, \ldots, \overline{i_{\ell-1}}\right) = c_0(i_1, \ldots, i_\ell) = M_f(i_1, \ldots, i_\ell) = f\left(\overline{i_1}, \ldots, \overline{i_\ell}\right).$$

Hence for any $i \in [N]$, the polynomials $\hat{f}_{1,i}(x_1, \ldots, x_{n\cdot(\ell-1)})$ and $\hat{f}(x_1, \ldots, x_{n\cdot(\ell-1)}, \bar{i})$ are two multilinear polynomials over $n \cdot (\ell - 1)$ variables that agree on all points of $\{0,1\}^{n\cdot(\ell-1)}$, and so they are the same polynomial, and in particular,

$$w_1(i) = \hat{f}_{1,i}(r_1, \ldots, r_{n\cdot(\ell-1)}) = \hat{f}(r_1, \ldots, r_{n\cdot(\ell-1)}, \bar{i}).$$

Prover input: A point $r \in \mathcal{M}_m$, a value $v \in \mathbb{F}_m$, and a codeword $c = (C_N)^{\otimes \ell}(M_f) \in (\mathbb{F}_1)^{(N')^\ell}$ for some function $f : \{0,1\}^m \to \mathbb{F}_1$

Verifier input: The point $r \in \mathcal{M}_m$ and the value $v \in \mathbb{F}_m$

1. Set $v_0 := v$ and $c_0 := c$.
2. For $t = 1, 2, \ldots, \ell$:
   ▷ **Invariant**: At the beginning of iteration t, a codeword $c_{t-1} \in (C_N)^{\otimes(\ell-(t-1))}$ and scalar $v_{t-1} \in \mathbb{F}_{n\cdot(\ell-(t-1))}$ have already been computed (where we set $\mathbb{F}_0 := \mathbb{F}_1$).
   (a) For every $i \in [N']$:
      i. Let $f_{t,i} : \{0,1\}^{n\cdot(\ell-t)} \to \mathbb{F}_1$ be the function given by
      $$f_{t,i}(\overline{i_1}, \ldots, \overline{i_{\ell-t}}) = c_{t-1}(i_1, \ldots, i_{\ell-t}, i)$$
      for any $i_1, \ldots, i_{\ell-t} \in [N]$.
      ii. P computes $\hat{f}_{t,i}(r_1, \ldots, r_{n\cdot(\ell-t)})$ using the algorithm given by Lemma 3.9 (cf., Fig. 2), where $\hat{f}_{t,i} : (\mathbb{F}_{n\cdot(\ell-t)})^{n\cdot(\ell-t)} \to \mathbb{F}_{n\cdot(\ell-t)}$ denotes the multilinear extension of $f_{t,i}$, when viewed as a function $f_{t,i} : \{0,1\}^{n\cdot(\ell-t)} \to \mathbb{F}_{n\cdot(\ell-t)}$; Let $w_t(i) \in \mathbb{F}_{n\cdot(\ell-t)}$ denote the output of the algorithm.
   (b) P sends $w_t \in (\mathbb{F}_{n\cdot(\ell-t)})^{N'}$ to V.
   (c) For every $b \in B_{n\cdot(\ell-t)}$, the verifier V checks that $w_t|_b \in \mathbb{F}_1^{N'}$ is a codeword of C_N, otherwise it rejects.
   (d) Let $y_t \in (\mathbb{F}_{n\cdot(\ell-t)})^N$ denote the systematic part of $w_t \in (\mathbb{F}_{n\cdot(\ell-t)})^{N'}$. View y_t as a function $y_t : \{0,1\}^n \to \mathbb{F}_{n\cdot(\ell-(t-1))}$ (by identifying an element $i \in [N]$ with its binary representation $\overline{i} \in \{0,1\}^n$, and recalling that $\mathbb{F}_{n\cdot(\ell-t)} \subseteq \mathbb{F}_{n\cdot(\ell-(t-1))}$), and let $\hat{y}_t : (\mathbb{F}_{n\cdot(\ell-(t-1))})^n \to \mathbb{F}_{n\cdot(\ell-(t-1))}$ denote the multilinear extension of y_t.
   V checks that $\hat{y}_t(r_{n\cdot(\ell-t)+1}, \ldots, r_{n\cdot(\ell-(t-1))}) = v_{t-1}$, otherwise it rejects.
   (e) V randomly chooses $\rho_{\ell-(t-1)} \in [N']$ and sends it to P.
   (f) Let $c_t \in (C_N)^{\otimes(\ell-(t-1))}$ be defined as $c_t(i_1, \ldots, i_{\ell-t}) = c_{t-1}(i_1, \ldots, i_{\ell-t}, \rho_{\ell-(t-1)})$ for any $i_1, \ldots, i_{\ell-t} \in [N']$, and let $v_t := w_t(\rho_{\ell-(t-1)}) \in \mathbb{F}_{n\cdot(\ell-t)}$.
3. V outputs the point $\rho = (\rho_1, \ldots, \rho_\ell) \in (N')^\ell$ and the value v_ℓ.

Figure 3: Fast Code Switching for MLE

But this implies in turn that the polynomials $\hat{y}_1(x_1,\ldots,x_n)$ and $\hat{f}(r_1,\ldots,r_{n\cdot(\ell-1)},x_1,\ldots,x_n)$ are both multilinear polynomials over n variables that agree on all points of $\{0,1\}^n$, so they are the same polynomial, and in particular

$$\hat{y}_1(r_{n\cdot(\ell-1)+1},\ldots,r_{n\cdot\ell}) = \hat{f}(r_1,\ldots,r_{n\cdot\ell}).$$

Next fix $t \in \{1,2,\ldots,\ell-1\}$, we shall show that $\hat{y}_{t+1}(r_{n\cdot(\ell-(t+1))+1},\ldots,r_{n\cdot(\ell-t)}) = w_t(\rho_{\ell-(t-1)})$. For every $i_1,\ldots,i_{\ell-t} \in [N]$, it holds that

$$f_{t+1,i_{\ell-t}}\left(\overline{i_1},\ldots,\overline{i_{\ell-(t+1)}}\right) = c_t\left(i_1,\ldots,i_{\ell-t}\right) = c_{t-1}\left(i_1,\ldots,i_{\ell-t},\rho_{\ell-(t-1)}\right) = f_{t,\rho_{\ell-(t-1)}}\left(\overline{i_1},\ldots,\overline{i_{\ell-t}}\right).$$

Hence for any $i \in [N]$, the polynomials $\hat{f}_{t+1,i}(x_1,\ldots,x_{n\cdot(\ell-(t+1))})$ and $\hat{f}_{t,\rho_{\ell-(t-1)}}(x_1,\ldots,x_{n\cdot(\ell-(t+1))},\bar{i})$ are two multilinear polynomials over $n\cdot(\ell-(t+1))$ variables that agree on all points of $\{0,1\}^{n\cdot(\ell-(t+1))}$, and so they are the same polynomial, and in particular,

$$w_{t+1}(i) = \hat{f}_{t+1,i}(r_1,\ldots,r_{n\cdot(\ell-(t+1))}) = \hat{f}_{t,\rho_{\ell-(t-1)}}(r_1,\ldots,r_{n\cdot(\ell-(t+1))},\bar{i}).$$

But this implies in turn that the polynomials $\hat{y}_{t+1}(x_1,\ldots,x_n)$ and $\hat{f}_{t,\rho_{\ell-(t-1)}}(r_1,\ldots,r_{n\cdot(\ell-(t+1))},x_1,\ldots,x_n)$ are both multilinear polynomials over n variables that agree on all points of $\{0,1\}^n$, so they are the same polynomial, and in particular

$$\hat{y}_{t+1}(r_{n\cdot(\ell-(t+1))+1},\ldots,r_{n\cdot(\ell-t)}) = \hat{f}_{t,\rho_{\ell-(t-1)}}(r_1,\ldots,r_{n\cdot(\ell-t)}) = w_t(\rho_{\ell-(t-1)}).$$

□

Next assume that $\hat{f}(r) = v$, we shall show that in this case the verifier does not reject, and $c(\rho_1,\ldots,\rho_\ell) = v_\ell$.

For any $t \in [m_2]$ and $i \in [N']$, it is satisfied that $w_t(i) = \hat{f}_{t,i}(r_1,\ldots,r_{n\cdot(\ell-t)})$. By linearity of the multilinear extension transformation, we get that w_t is a linear combination of the codewords in C_N over the field $\mathbb{F}_{n\cdot(\ell-t)}$. Additionally, from the linearity of projection over base elements, we get that for every $b \in B_{n\cdot(\ell-t)}$ it happens that $w_t|_b$ is a linear combination of the codewords of C_N and hence the verifier will not reject at Step 2c. Moreover, by assumption that $\hat{f}(r) = v$, and by the above Claim 3.11, the verifier does not reject on Step 2d. Finally, note that

$$v_\ell = w_\ell(\rho_1) = c_{\ell-1}(\rho_1) = c(\rho_1,\ldots,\rho_\ell).$$

Soundness. Next assume that $\hat{f}(r) \neq v$. Fix a prover strategy P^*, and denote by $w_1^*, w_2^*, \ldots, w_\ell^*$ the messages that P^* sends in Step 2b. For $t \in [\ell]$, let y_t^* denote the systematic part of w_t^*. Soundness relies on the following claim.

Claim 3.12. *If for some $t \in \{1,2,\ldots,\ell-1\}$, it holds that $w_t(\rho_{\ell-(t-1)}) \neq w_t^*(\rho_{\ell-(t-1)})$, and V does not reject on Steps 2c and 2d in iteration $t+1$, then with probability at least $1-\frac{\varepsilon}{\ell}$ over the choice of $\rho_{\ell-t}$, it holds that $w_{t+1}(\rho_{\ell-t}) \neq w_{t+1}^*(\rho_{\ell-t})$.*

Proof. Fix $t \in \{1,2,\ldots,\ell-1\}$. Since the verifier does not reject on Step 2d of iteration $t+1$, we get that

$$\hat{y}_{t+1}^*(r_{n\cdot(\ell-(t+1))+1},\ldots,r_{n\cdot(\ell-t)}) = w_t^*(\rho_{\ell-(t-1)}).$$

On the other hand, by Claim 3.11,

$$\hat{y}_{t+1}(r_{n\cdot(\ell-(t+1))+1},\ldots,r_{n\cdot(\ell-t)}) = w_t(\rho_{\ell-(t-1)}).$$

Given the assumption of the claim, this implies in turn that

$$\hat{y}_{t+1}(r_{n\cdot(\ell-(t+1))+1},\ldots,r_{n\cdot(\ell-t)}) \neq \hat{y}^*_{t+1}(r_{n\cdot(\ell-(t+1))+1},\ldots,r_{n\cdot(\ell-t)}),$$

and so $w_{t+1} \neq w^*_{t+1}$. Therefore there exists $b \in B_{n\cdot(\ell-t)}$ such that $w_t|_b \neq w^*_t|_b$. Since V does not reject on Step 2c of iteration $t+1$, we further have that both $w_{t+1}|_b$ and $w^*_{t+1}|_b$ are codewords of C_N, and so they must differ on at least a $(1-\frac{\varepsilon}{\ell})$-fraction of the entries. Consequently, with probability at least $1-\frac{\varepsilon}{\ell}$ over the choice of $\rho_{\ell-t} \in [N']$, it holds that $w_{t+1}|_b(\rho_{\ell-t}) \neq w^*_{t+1}|_b(\rho_{\ell-t})$, and so also $w_{t+1}(\rho_{\ell-t}) \neq w^*_{t+1}(\rho_{\ell-t})$. □

Now, if the verifier rejects in any of the iterations then we are done. Hence we may assume that the verifier does not reject in any of the iterations, and we will show that in this case $c(\rho) \neq w^*_\ell(r_\ell)$ with probability at least $1-\varepsilon$ over the choice of $\rho_\ell, \rho_{\ell-1}, \ldots, \rho_1$.

By assumption that $\hat{f}(r) \neq v$ and by Claim 3.11, we have that $\hat{y}_1(r_{n\cdot(\ell-1)+1},\ldots,r_{n\cdot\ell}) \neq v_0$. On the other hand, by assumption that the verifier does not reject on Step 2d, we have that $\hat{y}^*_1(r_{n\cdot(\ell-1)+1},\ldots,r_{n\cdot\ell}) = v_0$. So we conclude that $w_1 \neq w^*_1$, and in particular $w_1|_b \neq w^*_1|_b$ for some $b \in B_{n\cdot(\ell-1)}$. By assumption that the verifier does not reject on Step 2c, we have that $w_1|_b$ and $w^*_1|_b$ are both codewords of C_N, and so they differ by at least a $(1-\frac{\varepsilon}{\ell})$-fraction of the entries. Consequently, with probability at least $1-\frac{\varepsilon}{\ell}$ over the choice of ρ_ℓ, we have that $w_1|_b(\rho_\ell) \neq w^*_1|_b(\rho_\ell)$, and so $w_1(\rho_\ell) \neq w^*_1(\rho_\ell)$. Applying Claim 3.12 iteratively, and using a union bound, we conclude that with probability at least $1-\varepsilon$ over the choice of $\rho_\ell, \rho_{\ell-1}, \ldots, \rho_1$, it holds that $w_\ell(\rho_1) \neq w^*_\ell(\rho_1)$. But since $w_\ell(\rho_1) = c_{\ell-1}(\rho_1) = c(\rho_1,\ldots,\rho_\ell)$, this implies in turn that $w^*_\ell(\rho_1) \neq c(\rho_1,\ldots,\rho_\ell)$.

Number of rounds is clearly $\ell = \lceil 1/\gamma \rceil = O(1)$. Next we analyze the communication and randomness complexity, and prover and verifier running time.

Communication and Randomness Complexity. For any $t \in [\ell]$, in round t the prover sends $w_t \in (\mathbb{F}_{n\cdot(\ell-t)})^{N'}$, where each element of $\mathbb{F}_t$ can be represented using $O(\log(|\mathbb{F}_t|))$ bits, so the communication complexity is $N' \cdot O(\log(\mathbb{F}_{n\cdot(\ell-t)})) \leq N' \cdot O(\log(|\mathbb{F}_m|))$. So the total communication complexity is at most

$$\begin{aligned}
\ell \cdot N' \cdot O(\log(|\mathbb{F}_m|)) &\leq \ell \cdot N \cdot \mathrm{poly}(1/\varepsilon) \cdot O(\log(|\mathbb{F}_m|)) \\
&= \ell \cdot 2^{m/\ell} \cdot \mathrm{poly}(1/\varepsilon) \cdot O(\log(a_m)) \\
&\leq 2^{\frac{\gamma}{2}\cdot m} \cdot \mathrm{poly}(1/\varepsilon) \cdot O(\log(m/\epsilon)) \\
&\leq 2^{\gamma m} \cdot \mathrm{poly}(1/\varepsilon).
\end{aligned}$$

Additionally, in every round t the verifier sends a single element in $[N']$ and hence the total randomness complexity is at most

$$\ell \cdot \log(N') \leq \ell \cdot \log\big(N \cdot \mathrm{poly}(1/\varepsilon)\big) \leq \ell \cdot \big(\log(N) + O(\log(1/\varepsilon))\big) \leq m + O(\log(1/\varepsilon)).$$

Prover Complexity. In Step 2(a)ii of round t, the prover runs the algorithm from Lemma 3.9 for N' times, on a function over $n \cdot (\ell - t)$ variables. From the complexity analysis of the algorithm, this can be done via a circuit of size $2^{n\cdot(\ell-t)} \cdot \text{polylog}(1/\varepsilon) \leq 2^{n\cdot(\ell-1)} \cdot \text{polylog}(1/\varepsilon)$. Thus, overall, the prover can be implemented as a Boolean circuit of size at most

$$\begin{aligned}
\ell \cdot N' \cdot \text{polylog}(1/\varepsilon) \cdot 2^{n\cdot(\ell-1)} &\leq \ell \cdot N \cdot \text{poly}(1/\varepsilon) \cdot 2^{n\cdot(\ell-1)} \\
&= \ell \cdot 2^n \cdot \text{poly}(1/\varepsilon) \cdot 2^{n\cdot(\ell-1)} \\
&= \ell \cdot 2^{n\cdot\ell} \cdot \text{poly}(1/\varepsilon) \\
&\leq 2^m \cdot \text{poly}(1/\varepsilon).
\end{aligned}$$

Verifier Complexity. In Step 2c of round t, the verifier first projects the string $w_t \in (\mathbb{F}_{n\cdot(\ell-t)})^{N'}$ over all basis elements in $B_{n\cdot(\ell-t)}$, which can be done in time $N' \cdot \text{polylog}(|\mathbb{F}_{n\cdot(\ell-t)}|) \leq N \cdot \text{poly}(1/\varepsilon) \cdot \text{polylog}(|\mathbb{F}_m|)$ by solving a system of linear equation. Then, the verifier checks that every projection is a codeword of C_N which, since C_N is a systematic code, can be done by re-encoding the word in overall time $N \cdot \text{poly}(1/\varepsilon)$. In Step 2d of iteration t, the verifier computes the multilinear extension of $y_t \in (\mathbb{F}_{n\cdot(\ell-t)})^N$ at a single point in $\mathbb{F}_{n\cdot(\ell-(t-1))}$, which can be done in time $\text{poly}(N, \log(|\mathbb{F}_{n\cdot(\ell-(t-1))}|)) \leq \text{poly}(N, \log(|\mathbb{F}_m|))$. So the total verifier complexity is at most

$$\text{poly}(N, 1/\varepsilon, \log(|\mathbb{F}_m|)) \leq \text{poly}(2^{m/\ell}, 1/\varepsilon, \log(m/\varepsilon)) \leq 2^{O(\gamma m)} \cdot \text{poly}(1/\varepsilon).$$

□

4 Inner product check in log star rounds

In this section, we prove our main technical result, an $O(\log^*(n))$-round interactive proof with a linear-size prover which reduces checking the inner product of a constant number of strings to checking the value of an entry in the encoding of these strings via a linear-time encodable code C.

Theorem 4.1 (Inner product check with log star rounds)**.** *For any $\epsilon \in (0, \frac{1}{2})$, and constants $\gamma > 0$ and $d \in \mathbb{N}$, there exist a field $\mathbb{F}_1$ of characteristic 2 and of size $\text{poly}(1/\varepsilon)$, and a family $\mathcal{C} = \{C_n : \mathbb{F}_1^n \to \mathbb{F}_1^{n'}\}_{n\in\mathbb{N}}$ of linear codes of constant relative distance that can be encoded by a Boolean circuit of size $n \cdot \text{poly}(1/\varepsilon)$, so that the following holds.*

There exists an $O(\log^*(n))$-round interactive protocol, where the prover gets as input a tuple of d codewords $c_1 = C_n(y_1), \ldots, c_d = C_n(y_d) \in C_n$, where $y_1, \ldots, y_d \in \mathbb{F}_1^n$, and a value $v \in \mathbb{F}_1$, and the verifier gets as input only the value v. At the end of the interaction, the verifier either rejects or outputs indices $i_1, \ldots, i_d \in [n']$ and values $u_1, \ldots, u_d \in \mathbb{F}_1$ such that:

- ***Completeness:*** *If $\sum_{i\in[n]} y_1(i) \cdots y_d(i) = v$, then when V interacts with P it outputs $i_1, \ldots, i_d \in [n']$ and $u_1, \ldots, u_d \in \mathbb{F}_1$ such that $c_j(i_j) = u_j$ for any $j \in [d]$.*
- ***Soundness:*** *If $\sum_{i\in[n]} y_1(i) \cdots y_d(i) \neq v$ then, for every P^*, when V interacts with P^*, with probability at least $1 - \varepsilon$, either V rejects or it outputs $i_1, \ldots, i_d \in [n']$ and $u_1, \ldots, u_d \in \mathbb{F}_1$ such that $c_j(i_j) \neq u_j$ for some $j \in [d]$.*

The protocol has communication complexity $n^\gamma \cdot \text{poly}(1/\varepsilon)$ and randomness complexity $O(\log(n)) + \log^*(n) \cdot O(\log(1/\varepsilon))$. The prover can be implemented as a Boolean circuit of size $n \cdot \text{poly}(1/\varepsilon)$, and

the verifier has running time $n^{O(\gamma)} \cdot \mathrm{poly}(1/\varepsilon)$. *Moreover, any code of* $\mathcal{C}$ *is a* $\lceil 1/\gamma \rceil$*-dimensional tensor product, and the indices* $i_1, \ldots, i_d$ *only depend on* $\mathcal{V}$*'s randomness.*

Using proof composition, the interactive proof given in the above theorem can be turned into an $O(\log^*(n))$-query IOP with a polylogarithmic-time verifier.

Corollary 4.2. *For any* $\epsilon \in (0, \frac{1}{2})$*, and constants* $\gamma > 0$ *and* $d \in \mathbb{N}$*, there exists a field* $\mathbb{F}_1$ *of characteristic* 2 *and of size* $\mathrm{poly}(1/\varepsilon)$*, and a family* $\mathcal{C} = \{C_n : \mathbb{F}_1^n \to \mathbb{F}_1^{n'}\}_{n \in \mathbb{N}}$ *of linear codes of constant relative distance that can be encoded by a Boolean circuit of size* $n \cdot \mathrm{poly}(1/\varepsilon)$*, so that the following holds.*

There exists an $O(\log^*(n))$*-round and* $(\log^*(n) \cdot O(\log(1/\epsilon)))$*-query* IOP*, where the prover gets as input a tuple of* d *codewords* $c_1 = C_n(y_1), \ldots, c_d = C_d(y_d)$*, where* $y_1, \ldots, y_d \in \mathbb{F}_1^n$*, and a value* $v \in \mathbb{F}_1$*, and the verifier gets implicit access to the codewords* $c_1, \ldots, c_d$*, and explicit access to the value* v*. At the end of the interaction, the verifier makes a single query to each codeword* c_i*, and either accepts or rejects, so that the following holds:*

- ***Completeness:*** *If* $\sum_{i \in [n]} y_1(i) \cdots y_d(i) = v$*, then when* V *interacts with* P *it accepts with probability* 1*.*
- ***Soundness:*** *If* $\sum_{i \in [n]} y_1(i) \cdots y_d(i) \neq v$ *then, for every* P^**, when* V *interacts with* P^**, it rejects with probability at least* $1 - \epsilon$*.*

The IOP *has communication complexity* $n^{\gamma} \cdot \mathrm{poly}(1/\varepsilon)$*. The prover can be implemented as a Boolean circuit of size* $n \cdot \mathrm{poly}(1/\varepsilon)$*, and the verifier has running time* $\mathrm{polylog}(n/\epsilon)$*. Moreover, any code of* $\mathcal{C}$ *is a tensor product of dimension at least* $\lceil 1/\gamma \rceil$*.*

To prove Theorem 4.1, we first set in Section 4.1 below the parameters that will be used throughout this section. Then, in Section 4.2, we present an $O(\log^* n)$-round interactive proof with a linear-size prover which reduces checking the inner product of a constant number of strings to checking the evaluation of their *low-degree extension* at a special point coming from a *Matryoshka series*. In Section 4.3, we show a linear-time algorithm for computing the evaluation of the low-degree extension at Matryoshka points, and we use this algorithm in Section 4.4 to design a constant-round interactive protocol with a linear-size prover which reduces checking the evaluation of the low-degree extension at a Matryoshka point to checking the value of an entry in the encoding of the input strings via a linear-time encodable (tensor) code.

The above Theorem 4.1 then follows as a direct corollary of Lemma 4.5 from Section 4.2 and Lemma 4.11 from Section 4.4. In Section 4.5, we deduce Corollary 4.2 from Theorem 4.1 using proof composition.

4.1 Setting of parameters

Our goal is to design a protocol that given d strings $y_1, \ldots, y_d \in \mathbb{F}_1^n$ and a value $v \in \mathbb{F}_1$, checks that $\sum_{i \in [n]} y_1(i) \cdots y_d(i) = v$. In this section, it will be convenient for us to view each string y_j as (the truth table of) a function $f_j : H_1 \times H_2 \times \cdots \times H_m \to \mathbb{F}_1$ for $m = O(\log^*(n))$, and carefully chosen domains $H_1, \ldots, H_m$ satisfying that $\prod_{t=1}^{m} |H_t| = n$. Under this notation, our goal is to check that $\sum_{h \in H_1 \times \cdots \times H_m} f_1(h) \cdots f_d(h) = v$. Next we specify our choice of the domains $H_1, \ldots, H_m$.

Let σ_0, σ_1 be sufficiently large constants to be determined later on, let $\varepsilon > 0$ be a parameter, and let $\gamma > 0$ and $d \in \mathbb{N}$ be constants. We start by specifying the number m of the H_i's and their

sizes. Roughly speaking, we shall set $m = m_1 + m_2$ for $m_1 = \Theta(\log^* n)$ and $m_2 = \Theta(1/\gamma)$. The initial sequence of m_1 sets $H_1, \ldots, H_{m_1}$ will be chosen to be (sub-)exponentially increasing so that $|H_1| \cdots |H_{m_1}| \approx \text{polylog}(n)$, while the next m_2 sets $H_{m_1+1}, \ldots, H_m$ will be chosen to have the same size, under the constraint that $|H_1| \cdots |H_m| = n$.

More formally, we first define a series of exponentially-increasing integers $\{\ell_t\}_{t\in\mathbb{N}}$, by letting ℓ_1 be a sufficiently large integer to be determined soon, and for $t > 1$,

$$\ell_t := 2^{(\ell_{t-1})^{1/(2\sigma_0)}}. \tag{4}$$

Claim 4.3. *For any sufficiently large t, it holds that $\ell_t \geq (t+1)^3$.*

Proof. We prove this by induction, we start with the induction step and then derive the constraint on the base. Assume $\ell_t \geq (t+1)^3$, then by the definition of the series:

$$\ell_{t+1} = 2^{(\ell_t)^{1/(2\sigma_0)}} \geq 2^{(t+1)^{3/(2\sigma_0)}} \geq (t+2)^3,$$

where the last inequality holds for a sufficiently large t since the series $\frac{(t+2)^3}{2^{(t+1)^{3/(2\sigma_0)}}}$ converges to zero for a constant σ_0. Let t_0 be the minimal t for which the inequality holds. We choose ℓ_1 to be a sufficiently large constant so that at $\ell_{t_0} \geq (t_0+1)^3$, so the induction basis holds. □

Let m_1 be the minimal integer so that $\prod_{t=1}^{m_1} \ell_t \geq \log^{\sigma_0}(n)$, and note that

$$\ell_{m_1} = 2^{(\ell_{m_1-1})^{1/(2\sigma_0)}} \leq 2^{\sqrt{\log n}}. \tag{5}$$

Claim 4.4. *We have that $m_1 \leq 2 \cdot \log^*(n)$.*

Proof. For every σ_0, from some N on, $2^{k^{1/(2\cdot\sigma_0)}} \geq k^{2\cdot\sigma_0}$ for every $k \geq N$. Hence from some point on, given that ℓ_t is an increasing sequence,

$$\ell_{t+1} = 2^{(\ell_t)^{1/(2\sigma_0)}} = 2^{\left(2^{(\ell_{t-1})^{1/(2\cdot\sigma_0)}}\right)^{1/(2\cdot\sigma_0)}} \geq 2^{\left((\ell_{t-1})^{2\cdot\sigma_0}\right)^{1/2\cdot\sigma_0}} = 2^{\ell_{t-1}}.$$

We set $k_t = \ell_{2t-1}$, then $k_t \geq 2^{k_{t-1}}$ and hence $k_{t-1} \leq \log(k_t)$. Assume $k_{m_3} = \ell_{m_2} = \log^{\sigma_0}(n) \leq n$, and $k_1 = \ell_1$ is some constant, then by the definition of $\log^*$, we get $m_3 \leq \log^*(n)$. Taking into account the relation between ℓ_t and k_t, we get $m_2 \leq 2 \cdot \log^*(n)$.

□

Let $m_2 = \lceil 2/\gamma \rceil$ and $m = m_1 + m_2$. For $t \in [m]$, we choose the size n_t of H_t to be $n_t = \ell_t$ for $t \leq m_1$, and $n_t = \left(\frac{n}{\ell_1 \ell_2 \cdots \ell_{m_1}}\right)^{1/m_2}$ for $t > m_1$. Note that the n_t's are non-decreasing, that $n_1 \cdot n_2 \cdots n_m = n$, and that $m = O(\log^*(n))$, In particular, by assumption that γ is a constant, we have that $m \leq 2 \cdot \log^*(n) + O(1)$.

For $t \in [m]$, we shall choose H_t to be an arbitrary subset of size ℓ_t of a finite field $\mathbb{F}_t$, where $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$ is an increasing field ensemble of characteristic 2. To make sure that $\mathbb{F}$ has these properties, we first define a series $\{a_t\}_{t\in\mathbb{N}}$ of increasing powers of 2, by letting a_t be the smallest integer of the form 2^{2^j} that is larger than $\frac{d\cdot n_t \cdot t^2}{(\varepsilon\gamma)^{\sigma_1}}$. Note that for any $t \in \mathbb{N}$,

$$\frac{d \cdot n_t \cdot t^2}{(\varepsilon\gamma)^{\sigma_1}} < a_t \leq \left(\frac{d \cdot n_t \cdot t^2}{(\varepsilon\gamma)^{\sigma_1}}\right)^2. \tag{6}$$

For $t \in [m]$, let $\mathbb{F}_t = \text{GF}(a_t)$. Note that the field ensemble $\mathbb{F} = (\mathbb{F}_t)_{t\in\mathbb{N}}$ is an increasing field ensemble of characteristic 2. Let $\mathcal{M} := \mathcal{M}^{\mathbb{F}} = (\mathcal{M}_t)_{t\in\mathbb{N}}$ denote its Matryoshka series (cf., Definition 3.4).

4.2 Reducing Inner Product Claim to Matryoshka points in LDE

In this section, we present an $O(\log^*(n))$-round interactive protocol with a linear-size prover which reduces checking the inner product of d functions $f_1, \ldots, f_d : H_1 \times \cdots \times H_m \to \mathbb{F}_1$ to checking the evaluation of their *low-degree extensions*, at a special point coming from the Matryoshka series defined in the previous section.

Lemma 4.5. *Let $\varepsilon \in (0, \frac{1}{2})$ be a parameter, let $\gamma > 0$ and $d \in \mathbb{N}$ be constants, and let m, $\mathbb{F}_t$, $\mathcal{M}_t$, and H_t be defined as in Section 4.1 with respect to ε, γ, and d. Then there exists an $(m+1)$-round interactive protocol, where the prover gets as input a tuple of d functions $f_1, \ldots, f_d : H_1 \times \cdots \times H_m \to \mathbb{F}_1$ and a value $v \in \mathbb{F}_1$, and the verifier gets as input only the value v. At the end of the interaction, the verifier either rejects or outputs a point $r \in \mathcal{M}_m$ and values $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_m$ such that:*

- ***Completeness:*** *If $\sum_{h \in H_1 \times \cdots \times H_m} f_1(h) \cdots f_d(h) = v$, then when V interacts with P it outputs $r \in \mathcal{M}_m$ and $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_m$ such that $\hat{f}_j(r) = \alpha_j$ for any $j \in [d]$, where $\hat{f}_j : (\mathbb{F}_m)^m \to \mathbb{F}_m$ denotes the low-degree extension of f_j, when viewed as a function $f_j : H_1 \times \cdots \times H_m \to \mathbb{F}_m$.*
- ***Soundness:*** *If $\sum_{h \in H_1 \times \cdots \times H_m} f_1(h) \cdots f_d(h) \neq v$, then for every P^*, when V interacts with P^*, with probability at least $1 - \varepsilon$, either V rejects or it outputs $r \in \mathcal{M}_m$ and $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_m$ so that $\hat{f}_j(r) \neq \alpha_j$ for some $j \in [d]$.*

The protocol has communication complexity $n^\gamma \cdot O(\log(1/\varepsilon))$ and randomness complexity $O(\log(n)) + \log^*(n) \cdot O(\log(1/\varepsilon))$. The prover can be implemented as a Boolean circuit of size $n \cdot \mathrm{polylog}(1/\varepsilon)$, and the verifier has running time $n^\gamma \cdot \mathrm{polylog}(1/\varepsilon)$.

Proof. The protocol establishing Lemma 4.5 is very similar to the one given in Figure 1, where the main difference is that the domain of the functions are now the H_t's instead of just $\{0,1\}$, and to handle this we use the low-degree extension (LDE) instead of the multilinear extension (MLE). The formal description of the protocol is given in Fig. 4. We proceed to show that it satisfies the requirements.

Completeness. We start by proving the following claim:

Claim 4.6. *For any $j \in [d]$, $t \in \{0, 1, \ldots, m\}$, and $(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m$ it holds that*

$$f_{t,j}(h_{t+1}, \ldots, h_m) = \hat{f}_j(r_1, \ldots, r_t, h_{t+1}, \ldots, h_m).$$

Proof. Fix $j \in [d]$, the proof is by induction over t. For $t = 0$, this is true by our setting of $f_{0,j} = f_j$. Assuming the claim holds for $t-1$, we prove it holds for t. Fix $(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m$, we shall show that $f_{t,j}(h_{t+1}, \ldots, h_m) = \hat{f}_j(r_1, \ldots, r_t, h_{t+1}, \ldots, h_m)$. By definition of $f_{t,j}$ in Step 2f, we have that $f_{t,j}(h_{t+1}, \ldots, h_m) = \hat{f}_{t-1,j}(r_t, h_{t+1}, \ldots, h_m)$. On the other hand, by the induction hypothesis we have that for any $h \in H_t$,

$$\hat{f}_{t-1,j}(h, h_{t+1}, \ldots, h_m) = f_{t-1,j}(h, h_{t+1}, \ldots, h_m) = \hat{f}_j(r_1, \ldots, r_{t-1}, h, h_{t+1}, \ldots, h_m).$$

[8] We assume without loss of generality that n is a power of 2, otherwise padding with zeros at most doubles the input size.

Prover Input: A tuple of functions $f_1, \ldots, f_d : H_1 \times \cdots \times H_m \to \mathbb{F}_1$, a value $v \in \mathbb{F}_1$.
Verifier Input: The same value $v \in \mathbb{F}_1$.

The Protocol:

1. Set $v_0 := v$ and $f_{0,j} := f_j$ for every $j \in [d]$.
2. For $t = 1, 2, \ldots, m$:
   ▷ **Invariant**: *At the beginning of iteration t, the functions $f_{t-1,1}, \ldots, f_{t-1,d} : H_t \times \cdots \times H_m \to \mathbb{F}_{t-1}$ and scalar $v_{t-1} \in \mathbb{F}_{t-1}$ have already been computed (where we set $\mathbb{F}_0 := \mathbb{F}_1$). Recalling that $\mathbb{F}_{t-1}$ is a subfield of $\mathbb{F}_t$, we view the range of $f_{t-1,1}, \ldots, f_{t-1,d}$ as $\mathbb{F}_t$.*
   (a) Let $\Lambda_t \subseteq \mathbb{F}_t$ be an arbitrary subset of $d \cdot n_t$ distinct field elements such that $H_t \subseteq \Lambda_t$.
   (b) The prover P computes and sends to V the function $w_t : \Lambda_t \to \mathbb{F}_t$ defined as
   $$w_t(\lambda) = \sum_{(h_{t+1},\ldots,h_m) \in H_{t+1} \times \cdots \times H_m} \hat{f}_{t-1,1}(\lambda, h_{t+1}, \ldots, h_m) \cdots \hat{f}_{t-1,d}(\lambda, h_{t+1}, \ldots, h_m),$$
   where $\hat{f}_{t-1,j} : (\mathbb{F}_t)^{m-(t-1)} \to \mathbb{F}_t$ denotes the low-degree extension of $f_{t-1,j} : H_t \times \cdots \times H_m \to \mathbb{F}_t$.
   The computation is done as follows. For $j \in [d]$ and $h \in H_{t+1} \times \cdots \times H_m$, let $\hat{f}_{t-1,j,h} : \mathbb{F}_t \to \mathbb{F}_t$ denote the univariate extension of the function $f_{t-1,j,h} : H_t \to \mathbb{F}_t$, defined as $f_{t-1,j,h}(h_t) = f_{t-1,j}(h_t, h)$ for any $h_t \in H_t$, and note that $\hat{f}_{t-1,j}(\lambda, h) = \hat{f}_{t-1,j,h}(\lambda)$ for any $\lambda \in \mathbb{F}_t$ (since both $\hat{f}_{t-1,j}(\ldots, h)$ and $\hat{f}_{t-1,j,h}$ are univariate polynomials over $\mathbb{F}_t$ of degree at most $|H_t| - 1$ which agree on all points of H_t, and so they are the same polynomial). Consequently, for any $j \in [d]$ and $h \in H_{t+1} \times \cdots \times H_m$, one can compute the value $\hat{f}_{t-1,j}(\lambda, h) = \hat{f}_{t-1,j,h}(\lambda)$ for all $\lambda \in \Lambda$ using Theorem 2.9.
   (c) V checks that $\sum_{\lambda \in H_t} w_t(\lambda) = v_{t-1}$, otherwise it rejects.
   (d) V randomly chooses $r_t \in \mathbb{F}_t$ and sends it to P.
   (e) V computes $v_t = \hat{w}_t(r_t)$, where $\hat{w}_t : \mathbb{F}_t \to \mathbb{F}_t$ denotes the univariate extension of $w_t : \Lambda_t \to \mathbb{F}_t$ (a degree $d \cdot n_t - 1$ polynomial).
   (f) For any $j \in [d]$, P computes $f_{t,j} : H_{t+1} \times \cdots \times H_m \to \mathbb{F}_t$, defined as $f_{t,j}(h_{t+1}, \ldots, h_m) = \hat{f}_{t-1,j}(r_t, h_{t+1}, \ldots, h_m)$ for all $h_{t+1}, \ldots, h_m \in H_{t+1} \times \cdots \times H_m$. as described in Step 2b.
3. P sends $\alpha_1 = f_{m,1}, \ldots, \alpha_d = f_{m,d} \in \mathbb{F}_m$.
4. V checks that $\alpha_1 \cdots \alpha_d = v_m$, otherwise it rejects.
5. V outputs $r = (r_1, \ldots, r_m) \in \mathcal{M}_m$ and $\alpha_1, \ldots, \alpha_d \in \mathbb{F}_m$.

Figure 4: Matroyshka Inner Product Check in Log-star Rounds

So $\hat{f}_{t-1,j}(x, h_{t+1}, \ldots, h_m)$ and $\hat{f}_j(r_1, \ldots, r_{t-1}, x, h_{t+1}, \ldots, h_m)$ are two univariate polynomials over $\mathbb{F}_m$ in the indeterminate x of degree at most $|H_t| - 1$ that agree on all values in H_t, and so these polynomials must be identical. We conclude that

$$f_{t,j}(h_{t+1}, \ldots, h_m) = \hat{f}_{t-1,j}(r_t, h_{t+1}, \ldots, h_m) = \hat{f}_j(r_1, \ldots, r_t, h_{t+1}, \ldots, h_m).$$

□

Completeness relies on the following claim.

Claim 4.7. *For any $t \in \{1, 2, \ldots, m-1\}$, it holds that $\sum_{\lambda \in H_{t+1}} w_{t+1}(\lambda) = \hat{w}_t(r_t)$.*

Proof. Fix $t \in \{1, 2, \ldots, m-1\}$. Then we have that

$$\begin{aligned}
\sum_{\lambda \in H_{t+1}} w_{t+1}(\lambda) &= \sum_{\lambda \in H_{t+1}} \sum_{(h_{t+2}, \ldots, h_m) \in H_{t+2} \times \cdots \times H_m} \hat{f}_{t,1}(\lambda, h_{t+2} \ldots, h_m) \cdots \hat{f}_{t,d}(\lambda, h_{t+2}, \ldots, h_m) \\
&= \sum_{(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m} f_{t,1}(h_{t+1}, \ldots, h_m) \cdots f_{t,d}(h_{t+1}, \ldots, h_m) \\
&= \sum_{(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m} \hat{f}_{t-1,1}(r_t, h_{t+1}, \ldots, h_m) \cdots \hat{f}_{t-1,d}(r_t, h_{t+1}, \ldots, h_m) \\
&= \widehat{w}_t(r_t),
\end{aligned}$$

where the last equality follows since $\widehat{w}_t(x)$ and

$$\sum_{(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m} \hat{f}_{t-1,1}(x, h_{t+1}, \ldots, h_m) \cdots \hat{f}_{t-1,d}(x, h_{t+1}, \ldots, h_m)$$

are both polynomials of degree at most $d \cdot n_t - 1$ in the indeterminate x, which by Step 2b, agree on all $d \cdot n_t$ points $\lambda \in \Lambda$, hence they must be the same polynomial. □

Next assume that $\sum_{h \in H_1 \times \cdots \times H_m} f_1(h) \cdots f_d(h) = v$. Then we have that

$$\begin{aligned}
\sum_{\lambda \in H_1} w_1(\lambda) &= \sum_{\lambda \in H_1} \sum_{(h_2, \ldots, h_m) \in H_2 \times \cdots \times H_m} \hat{f}_{0,1}(\lambda, h_2 \ldots, h_m) \cdots \hat{f}_{0,d}(\lambda, h_2, \ldots, h_m) \\
&= \sum_{h \in H_1 \times \cdots \times H_m} f_1(h) \cdots f_d(h) = v_0,
\end{aligned}$$

and so the verifier does not reject on Step 2c of the first iteration. Moreover, by Claim 4.7 above, the verifier also does not reject on Step 2c in any of the iterations $t \in \{2, \ldots, m\}$. Furthermore, we have that

$$\alpha_1 \cdots \alpha_d = f_{m,1} \cdots f_{m,d} = \hat{f}_{m-1,1}(r_m) \cdots \hat{f}_{m-1,d}(r_m) = \widehat{w}_m(r_m) = v_m,$$

where the third equality follows by the same argument as in the proof of Claim 4.7, and so the verifier does not reject on Step 4 as well. Finally, by Claim 4.6, the verifier outputs $\alpha_1, \ldots, \alpha_d$, and r which satisfy that $\alpha_j = f_{m,j} = \hat{f}_j(r)$ for any $j \in [d]$.

Soundness. Next assume that $\sum_{h\in H_1\times\cdots\times H_m} f_1(h)\cdots f_d(h) \neq v$. Fix a prover strategy P^*, and denote by $w_1^*, w_2^*, \ldots, w_m^*$ the messages that P^* sends in Step 2b. Soundness relies on the following claim.

Claim 4.8. *If for some $t \in \{1,2,\ldots,m-1\}$, it holds that $\hat{w}_t^*(r_t) \neq \hat{w}_t(r_t)$, and V does not reject in Step 2c in iteration $t+1$, then with probability at least $1 - \frac{d\cdot n_{t+1}}{|\mathbb{F}_{t+1}|}$ over the choice of r_{t+1}, it holds that $\hat{w}_{t+1}^*(r_{t+1}) \neq \hat{w}_{t+1}(r_{t+1})$.*

Proof. Fix $t \in \{1,\ldots,m-1\}$, and assume that $\hat{w}_t^*(r_t) \neq \hat{w}_t(r_t)$, and that V does not reject in Step 2c in iteration $t+1$. By assumption that V does not reject in Step 2c of iteration $t+1$, we have that $\sum_{\lambda\in H_t} w_{t+1}^*(\lambda) = \widehat{w}_t^*(r_t)$. On the other hand, by Claim 4.7 we know that $\sum_{\lambda\in H_t} w_{t+1}(\lambda) = \widehat{w}_t(r_t)$. Therefore, by assumption that $\hat{w}_t^*(r_t) \neq \hat{w}_t(r_t)$, we have that $\sum_{\lambda\in H_t} w_{t+1}^*(\lambda) \neq \sum_{\lambda\in H_t} w_{t+1}(\lambda)$. Hence $\widehat{w}_{t+1}$ and $\widehat{w}_{t+1}^*$ are two distinct univariate polynomials of degree at most $d\cdot n_{t+1} - 1$ over $\mathbb{F}_{t+1}$, and so they agree on at most a $\frac{d\cdot n_{t+1}}{|\mathbb{F}_{t+1}|}$-fraction of the points. Thus with probability at least $1 - \frac{d\cdot n_{t+1}}{|\mathbb{F}_{t+1}|}$ over the choice of r_{t+1}, it holds that $\widehat{w}_{t+1}(r_{t+1}) \neq \widehat{w}_{t+1}^*(r_{t+1})$. □

Now, if the verifier rejects in any of the iterations then we are done. Hence we may assume that the verifier does not reject in any of the iterations.

By assumption that $\sum_{h\in H_1\times\cdots\times H_m} f_1(h)\cdots f_d(h) \neq v$, we have that

$$\sum_{\lambda\in H_1} w_1(\lambda) = \sum_{h\in H_1\times\cdots\times H_m} f_1(h)\cdots f_d(h) \neq v_0.$$

On the other hand, by assumption that the verifier does not reject on Step 2c, we also have that $\sum_{\lambda\in H_1} w_1^*(\lambda) = v_0$. We conclude that $\hat{w}_1$ and $\hat{w}_1^*$ are two distinct univariate polynomials of degree at most $d\cdot n_1 - 1$, and by the same argument as in the proof of Claim 4.8, this implies in turn that $\hat{w}_1(r_1) \neq \hat{w}_1^*(r_1)$ with probability at least $1 - \frac{d\cdot n_1}{|\mathbb{F}_1|}$ over the choice of r_1. Applying Claim 4.8 above iteratively, and using the union bound, we then conclude that with probability at least $1 - \sum_{t=1}^{m} \frac{d\cdot n_t}{|\mathbb{F}_t|}$, it holds that $\hat{w}_m(r_m) \neq \hat{w}_m^*(r_m)$.

Next assume that this latter event holds, i.e., that $\hat{w}_m(r_m) \neq \hat{w}_m^*(r_m)$, and let $\alpha_1^*, \ldots, \alpha_d^*$ denote the messages sent by P on Step 4. Then assuming that the verifier does not reject on Step 4, we have that $\alpha_1^*\cdots\alpha_d^* = \hat{w}_m^*(r_m)$. On the other hand, by Claim 4.6 we have that

$$\hat{w}_m(r_m) = f_{m,1}\cdots f_{m,d} = \hat{f}_1(r)\cdots\hat{f}_d(r).$$

We conclude that in this case there exists $j \in [d]$ such that $\hat{f}_j(r) \neq \alpha_j^*$.

Overall, we obtain a soundness error of at most

$$\sum_{t=1}^{m} \frac{n\cdot d_t}{|\mathbb{F}_t|} \leq \sum_{t=1}^{\infty} \frac{n\cdot d_t}{|\mathbb{F}_t|} = \sum_{t=1}^{\infty} \frac{n\cdot d_t}{a_t} \leq \sum_{t=1}^{\infty} \frac{(\varepsilon\gamma)^{\sigma_0}}{t^2} \leq \varepsilon^{\sigma_1}\cdot\sum_{t=1}^{\infty}\frac{1}{t^2} < \varepsilon,$$

where the last inequality follows for a sufficiently large constant σ_1.

Number of rounds is clearly $m+1$. Next we analyze the communication and randomness complexity, and prover and verifier running time.

Randomness Complexity. In every round t, the verifier sends a single random elements of $\mathbb{F}_t$, where each field element can be represented using $O(\log(|\mathbb{F}_t|))$ bits. Hence there exists an absolute constant $\xi_0 > 0$ so that the total randomness complexity is at most

$$\begin{aligned}\sum_{t=1}^{m} \xi_0 \cdot \log(|\mathbb{F}_t|) &= \sum_{t=1}^{m} \xi_0 \cdot \log(a_t) \\ &\leq m_1 \cdot O(\log(a_{m_1})) + m_2 \cdot O(\log(a_m)) \\ &\leq m_1 \cdot O(\log(n_{m_1} \cdot m/\varepsilon)) + m_2 \cdot O(\log(n_m \cdot m/\varepsilon)) \\ &\leq m_1 \cdot O(\log(\ell_{m_1})) + m_2 \cdot O(\log(n^{1/m_2})) + m \cdot O(\log(m/\varepsilon)) \\ &\leq O(m_1\sqrt{\log n}) + O(\log n) + m \cdot O(\log(m/\varepsilon)) \\ &\leq O(\log n) + \log^*(n) \cdot O(\log(1/\varepsilon)).\end{aligned}$$

Communication complexity. In every round t, the prover sends $d \cdot n_t$ elements of the field $\mathbb{F}_t$, and so there exists an absolute constant $\xi_0 > 0$ so that the total communication is at most

$$\begin{aligned}\sum_{t=1}^{m} \xi_0 \cdot n_t \cdot \log(|\mathbb{F}_t|) &= \sum_{t=1}^{m} \xi_0 \cdot n_t \cdot \log(a_t) \\ &\leq m \cdot n_m \cdot O(\log(a_m)) \\ &\leq m \cdot n_m \cdot O(\log(n_m \cdot m/\varepsilon)) \\ &\leq m \cdot n^{1/m_2} \cdot O(\log(n^{1/m_2} \cdot m/\varepsilon)) \\ &\leq n^{2/m_2} \cdot O(\log(1/\varepsilon)) \\ &\leq n^{\gamma} \cdot O(\log(1/\varepsilon)).\end{aligned}$$

Prover complexity. In every round t, in Step 2b, the prover has to compute for any $j \in [d]$ and $(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m$, the value $\hat{f}_{t-1,j}(\lambda, h_{t+1}, \ldots, h_m)$ on all points $\lambda \in \Lambda$, which can be done using $\tilde{O}(n_t)$ field operations by Theorem 2.9. In Step 2f, the prover has to compute for any $j \in [d]$ and $(h_{t+1}, \ldots, h_m) \in H_{t+1} \times \cdots \times H_m$, an additional value $\hat{f}_{t-1,j}(r_t, h_{t+1}, \ldots, h_m)$, which can also be done using $\tilde{O}(n_t)$ field operations. Additionally, the prover has to use $O(n_{t+1} \cdots n_m)$ field operations in $\mathbb{F}_t$ to compute the inner product on Step 2b from these values.

So overall, in round t the verifier has to use $O(n_t \cdot n_{t+1} \cdots n_m) \cdot \text{polylog}(n_t)$ field operations over $\mathbb{F}_t$. Since each field operation in $\mathbb{F}_t$ can be performed in time $\text{polylog}(|\mathbb{F}_t|) = \text{polylog}(a_t)$, and recalling that $a_t \geq n_t$, overall there exists an absolute constant $\xi_0 > 0$ so that the prover can be implemented as a Boolean circuit of size:

$$\sum_{t=1}^{m} n_t \cdots n_m \cdot \log^{\xi_0}(a_t) = \sum_{t=1}^{m_1} \frac{n}{n_1 \cdots n_{t-1}} \cdot \log^{\xi_0}(a_t) + \sum_{t=m_1+1}^{m} \frac{n}{n_1 \cdots n_{t-1}} \cdot \log^{\xi_0}(a_t).$$

To bound the first summand, note that:

$$\begin{aligned}
\sum_{t=1}^{m_1} \frac{n}{n_1 \cdots n_{t-1}} \cdot \log^{\xi_0}(a_t) &\leq \sum_{t=1}^{m_1} \frac{n}{n_{t-2} \cdot n_{t-1}} \cdot \log^{2\xi_0}((t \cdot n_t)/\varepsilon) \\
&= \sum_{t=1}^{m_1} \frac{n}{\ell_{t-2} \cdot \ell_{t-1}} \cdot \log^{2\xi_0}((t \cdot \ell_t)/\varepsilon) \\
&\leq \sum_{t=1}^{m_1} \frac{n}{\ell_{t-2} \cdot \ell_{t-1}} \cdot \left(\log^{2\xi_0}(t/\varepsilon) + \log^{2\xi_0}(\ell_t)\right) \\
&\leq \sum_{t=1}^{m_1} \frac{n}{\ell_{t-2} \cdot \ell_{t-1}} \cdot \left(\log^{2\xi_0}(t/\varepsilon) + (\ell_{t-1})^{\xi_0/\sigma_0}\right) \\
&\leq \sum_{t=1}^{m_1} \frac{n}{\ell_{t-2}} \cdot \log^{2\xi_0}(t/\varepsilon) \\
&\leq \sum_{t=1}^{m_1} \frac{n}{t^3} \cdot \log^{2\xi_0}(t/\varepsilon) \\
&\leq n \cdot \mathrm{polylog}(1/\varepsilon),
\end{aligned}$$

where the third inequality follows by Eq. (4), the fourth inequality follows for $\sigma_0 \geq \xi_0$, and the one before last inequality follows by Claim 4.3.

Next we bound the second summand:

$$\begin{aligned}
\sum_{t=m_1+1}^{m} \frac{n}{n_1 \cdots n_{t-1}} \cdot \log^{\xi_0}(a_t) &\leq m_2 \cdot \frac{n}{n_1 \cdots n_{m_1}} \cdot \mathrm{polylog}((t \cdot n_t)/\varepsilon) \\
&\leq \frac{n}{\ell_1 \cdots \ell_{m_1}} \cdot \mathrm{polylog}(n/\varepsilon) \\
&\leq \frac{n}{\log^{\sigma_0} n} \cdot \mathrm{polylog}(n/\varepsilon) \\
&\leq n \cdot \mathrm{polylog}(1/\varepsilon),
\end{aligned}$$

where the one before last inequality follows by our choice of m_1, and the last inequality follows for a sufficiently large constant σ_0.

So overall, we get that the prover complexity is $n \cdot \mathrm{polylog}(1/\varepsilon)$.

Verifier complexity. In iteration t, in Step 2c, the verifier computes the sum of n_t elements in $\mathbb{F}_t$. In Step 2e, the verifier needs to compute the value of a polynomial of degree at most $d \cdot n_t$ in the field $\mathbb{F}_t$ given the values of $d \cdot n_t$ points, which can be done using $\tilde{O}(n_t)$ field operations over $\mathbb{F}_t$.

Overall, we get that there exists an absolute constant $\xi_0 > 0$ so that the verifier complexity is

at most

$$\begin{aligned}\sum_{t=1}^{m} n_t \cdot \log^{\xi_0}(|\mathbb{F}_t|) &= \sum_{t=1}^{m} n_t \cdot \log^{\xi_0}(a_t) \\ &\leq m \cdot n_m \cdot \text{polylog}(a_m) \\ &\leq m \cdot n_m \cdot \text{polylog}(n_m \cdot m/\varepsilon) \\ &\leq m \cdot n^{1/m_2} \cdot \text{polylog}(n^{1/m_2} \cdot m/\varepsilon) \\ &\leq n^{2/m_2} \cdot \text{polylog}(1/\varepsilon) \\ &\leq n^{\gamma} \cdot \text{polylog}(1/\varepsilon).\end{aligned}$$

□

4.3 Computing a Matryoshka point in the LDE

Next we show that the low-degree extension of a given function $f : H_1 \times \cdots \times H_m \to \mathbb{F}_1$ can be evaluated in *linear time* on points coming from the Matryoshka series corresponding to our choice of field ensemble. In fact, for our final protocol, we shall need a slightly stronger statement that applies to any $m' \leq m$.

Lemma 4.9. *Let $\varepsilon \in (0, \frac{1}{2})$ be a parameter, let $\gamma > 0$, and $d \in \mathbb{N}$ be constants, and let $\mathbb{F}_t$, $\mathcal{M}_t$, and H_t be defined as in Section 4.1 with respect to ε, γ, and d. Let $m' \in [m]$ be a parameter. Then there exists a Boolean circuit of size $\text{polylog}(1/\varepsilon) \cdot \prod_{t=1}^{m'} |H_i|$ that given as input a function $f : H_1 \times \cdots \times H_{m'} \to \mathbb{F}_1$ and a point $r \in \mathcal{M}_{m'}$, outputs $\hat{f}(r) \in \mathbb{F}_{m'}$, where $\hat{f}$ denotes the low-degree extension of f, when viewed as a function $f : H_1 \times \cdots \times H_{m'} \to \mathbb{F}_t$. In particular, when $m' = m$ the circuit size is $n \cdot \text{polylog}(1/\varepsilon)$, and when $m' < m$ the circuit size is at most $\frac{n}{|H_m|} \cdot \text{poly}(1/\varepsilon)$.*

Proof. The algorithm is presented in Fig. 5. We proceed by proving it.

Correctness.

Claim 4.10. *For every $t \in \{0, 1, \ldots, m'\}$, it holds that $f_t(h_{t+1}, \ldots, h_{m'}) = \hat{f}_{t-1}(r_1, \ldots, r_t, h_{t+1}, \ldots, h_{m'})$ for every $(h_{t+1}, \ldots, h_{m'}) \in H_t \times \cdots \times H_{m'}$.*

Proof. We prove this by induction over t. For the base case (i.e. $t = 0$) this follows from the definition of f_0 and of low degree extension,

$$f_0(h_1, \ldots, h_{m'}) = f(h_1, \ldots, h_{m'}) = \hat{f}(h_1, \ldots, h_{m'}).$$

Assuming the claim holds for $t-1$, we prove it holds for t. Let $(h_{t+1}, \ldots, h_{m'}) \in H_t \times \cdots \times H_{m'}$, then the polynomials $\hat{f}_{t-1}(*, h_{t+1}, \ldots, h_{m'})$ and $\hat{f}(r_1, \ldots, r_{t-1}, *, h_{t+1}, \ldots, h_{m'})$ are $(|H_t|-1)$-degree polynomials. From the assumption over $t-1$, these polynomials agree on all points in $|H_t|$, hence they agree on all of $\mathbb{F}_t$, specifically $r_t \in \mathbb{F}_t$,

$$f_t(h_{t+1}, \ldots, h_{m'}) = \hat{f}_{t-1}(r_t, h_{t+1}, \ldots, h_{m'}) = \hat{f}(r_1, \ldots, r_t, h_{t+1}, \ldots, h_{m'}).$$

□

Correctness follows immediately from the claim above wrt $t = m'$.

Input: A function $f : H_1 \times \cdots \times H_{m'} \to \mathbb{F}_1$ and a point $r = (r_1, \ldots, r_{m'}) \in \mathcal{M}_{m'}$.
Output: $\hat{f}(r)$.

The Algorithm:

1. Set $f_0 := f$.
2. For $t = 1, 2, \ldots, m'$:
   ▷ **Invariant**: *At the beginning of iteration t, the function $f_{t-1} : H_t \times \cdots \times H_{m'} \to \mathbb{F}_{t-1}$ has already been computed (where we set $\mathbb{F}_0 := \mathbb{F}_1$). Recalling that $\mathbb{F}_{t-1}$ is a subfield of $\mathbb{F}_t$, in what follows, we view the range of f_{t-1} as being $\mathbb{F}_t$.*

   Compute $f_t : H_{t+1} \times \cdots \times H_{m'} \to \mathbb{F}_t$, defined as $f_t(h_{t+1}, \ldots, h_{m'}) = \hat{f}_{t-1}(r_t, h_{t+1}, \ldots, h_{m'})$ for all $(h_{t+1}, \ldots, h_{m'}) \in H_{t+1} \times \cdots \times H_{m'}$, where $\hat{f}_{t-1}$ denotes the low-degree extension of $f_{t-1} : H_t \times \cdots \times H_{m'} \to \mathbb{F}_t$.
3. Output $f_{m'} \in \mathbb{F}_{m'}$.

Figure 5: Fast LDE Evaluation at Matryoshka Points

Complexity. For every t, in step 2, the prover computes the $|H_{t+1} \times \cdots \times H_{m'}|$ values of f_t. Each of the values is computed as the $(|H_t| - 1)$-degree extension of a previously computed function over the field $\mathbb{F}_t$, for each point this can be done through $O(n_t) \cdot \mathrm{polylog}(n_t)$ operations in $\mathbb{F}_t$ by Theorem 2.9.

Since these are constructible fields, operations in the field can be computed using boolean circuits of size $\mathrm{polylog}(|\mathbb{F}_t|) = \mathrm{polylog}(a_t)$. Overall, given that $a_t \geq n_t$, there exists an absolute constant ξ_0 such that the complexity is

$$\sum_{t=1}^{m'} |H_{t+1} \times \cdots \times H_{m'}| \cdot |H_t| \cdot \log^{\xi_0}(a_t)$$

Recalling that $n_t = |H_t|$ and letting $n' := \prod_{t=1}^{m'} n_t$, we get that the above expression is

$$\sum_{t=1}^{m'} \frac{n' \cdot \log^{\xi_0}(a_t)}{\prod_{i=1}^{t} n_i}$$

Similarly to the prover complexity analysis of Lemma 4.5, we get that the latter expression is at most $n' \cdot \mathrm{polylog}(1/\varepsilon)$.

□

4.4 Fast code switching for LDE

In this section, we use the algorithm from the previous section to design a constant-round interactive protocol with a linear-size prover which reduces checking the evaluation of the low-degree extension

of a given function $f : H_1 \times \cdots \times H_m \to \mathbb{F}_1$ at a Matryoshka point to checking the value of an entry in the encoding of (the truth table of) f via a linear-time encodable code.

Lemma 4.11. *Let $\varepsilon \in (0, \frac{1}{2})$ be a parameter, let $\gamma > 0$ and $d \in \mathbb{N}$ be constants, and let m, $\mathbb{F}_t$, $\mathcal{M}_t$, and H_t be defined as in Section 4.1 with respect to ε, γ, and d. Then there exists a systematic linear code ensemble $\mathcal{C} = \{(\mathbb{F}_1)^n \to (\mathbb{F}_1)^{n'}\}_{n \in \mathbb{N}}$ of constant relative distance that can be encoded using a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$, and a constant-round interactive protocol with the following properties:*

- ***Prover's input:*** *The prover gets as input a point $r \in \mathcal{M}_m$, a value $v \in \mathbb{F}_m$, and a codeword $c = \mathcal{C}(f)$ for some function $f : H_1 \times \cdots \times H_m \to \mathbb{F}_1$, where f is viewed as a binary string of length $n = |H_1| \cdots |H_m|$.*
- ***Verifier's input:*** *The verifier gets as input the point $r \in \mathcal{M}_m$ and the value $v \in \mathbb{F}_m$.*
- ***Completeness:*** *If $\hat{f}(r) = v$, then when V interacts with P, it outputs an entry ρ and a value $u \in \mathbb{F}_1$ so that $c(\rho) = u$, where $\hat{f}$ denotes the low-degree extension of f, when viewed as a function $f : H_1 \times \cdots \times H_m \to \mathbb{F}_m$.*
- ***Soundness:*** *If $\hat{f}(r) \neq v$, then for every P^*, when V interacts with P^*, with probability at least $1 - \varepsilon$, it either rejects or outputs ρ and $u \in \mathbb{F}_1$ so that $c(\rho) \neq u$.*

The protocol has communication complexity $n^\gamma \cdot \mathrm{poly}(1/\varepsilon)$ and randomness complexity $\log(n) + O(\log(1/\varepsilon))$. The prover can be implemented as a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$, and the verifier has running time $n^{O(\gamma)} \cdot \mathrm{poly}(1/\varepsilon)$. Moreover, any code of $\mathcal{C}$ is a $\lceil 1/\gamma \rceil$-dimensional tensor product, and the entry ρ only depends on $\mathcal{V}$'s randomness.

Proof. The protocol establishing Lemma 4.11 is very similar to the one given in Fig. 3, where the main difference is that the domain of the functions are now the H_t's instead of just $\{0,1\}$, and that we consider the low-degree extension (LDE) instead of the multilinear extension (MLE) of f.

The code ensemble $\mathcal{C}$: Let $\{C_n : (\mathbb{F}_1)^n \to (\mathbb{F}_1)^{n'}\}_{n \in \mathbb{N}}$ be the systematic linear code ensemble of relative distance $1 - \frac{\epsilon}{m_2+1}$ that is encodable using a Boolean circuit of size $n \cdot \mathrm{poly}(1/\varepsilon)$, guaranteed by Theorem 2.10 for a sufficiently large constant σ_1. For $n \in \mathbb{N}$, let $N_1 := \prod_{t=1}^{m_1} n_t$ and $N_2 := n_{m_2+1}$, and let $\mathcal{C} = \{C_{N_1} \otimes (C_{N_2})^{\otimes m_2}\}_{n \in \mathbb{N}}$. Note that by Fact 2.6, $C_{N_1} \otimes (C_{N_2})^{\otimes m_2}$ is a code of message length $n = N_1 \cdot (N_2)^{m_2}$ that can be encoded in time $n \cdot \mathrm{poly}(1/\varepsilon)$.

The encoding of a function $f : H_1 \times \cdots \times H_m \to \mathbb{F}_1$ via $\mathcal{C}$ is performed as follows. We view f as a string $M_f \in (\mathbb{F}_1)^{N_1 \times (N_2)^{m_2}}$ (noting that $N_1 \cdot (N_2)^{m_2} = n$) which satisfies that $M_f((i_1, \ldots, i_{m_1}), i_{m_1+1}, \ldots, i_m) = f(i_1, \ldots, i_m)$ for any $(i_1, \ldots, i_m) \in [n_1] \times \cdots \times [n_m]$, where we associate indices $i_t \in [n_t]$ with elements of H_t, and tuples $(i_1, \ldots, i_{m_1}) \in [n_1] \times \cdots \times [n_{m_1}]$ with elements of $[N_1]$. The encoding $\mathcal{C}(f)$ of f is $(C_{N_1} \otimes (C_{N_2})^{\otimes m_2})(M_f) \in (\mathbb{F}_1)^{N_1' \times (N_2')^{m_2}}$.

The interactive protocol: The formal description of the protocol is given in Fig. 6. In what follows, every t, we view $\mathbb{F}_t$ as a $\log_{a_1}(a_t)$-dimensional vector space over $\mathbb{F}_1$ (recalling that $\mathbb{F}_1 \subseteq \mathbb{F}_t$), and fix a basis B_t of the field $\mathbb{F}_t$ over $\mathbb{F}_1$. Every element $\alpha \in \mathbb{F}_t$ can be uniquely expressed as a linear combination $\alpha = \sum_{b \in B_t} \alpha|_b \cdot b$ where $\alpha|_b \in \mathbb{F}_1$. More generally, any vector $w \in (\mathbb{F}_t)^n$ can be uniquely expressed as a linear combination $w = \sum_{b \in B_t} w|_b \cdot b$, where $w|_b \in (\mathbb{F}_1)^n$.

We proceed to show that the protocol satisfies the requirements.

Prover input: A point $r \in \mathcal{M}_m$, a value $v \in \mathbb{F}_m$, and a codeword $c = (C_{N_1} \otimes (C_{N_2})^{\otimes m_2})(M_f) \in (\mathbb{F}_1)^{N_1' \times (N_2')^{m_2}}$ for a function $f : H_1 \times \cdots \times H_m \to \mathbb{F}_1$

Verifier input: The point $r \in \mathcal{M}_m$ and the value $v \in \mathbb{F}_m$.

1. Set $v_0 := v$ and $c_0 := c$.
2. For $t = 1, 2, \ldots, m_2$:
   ▷ **Invariant**: At the beginning of iteration t, a codeword $c_{t-1} \in C_{N_1} \otimes (C_{N_2})^{\otimes(m_2-(t-1))}$ and a scalar $v_{t-1} \in \mathbb{F}_{m-(t-1)}$ have already been computed.
   (a) For every $i \in [N_2']$:
      i. Let $f_{t,i} : H_1 \times \cdots \times H_{m-t} \to \mathbb{F}_1$ be the function given by
      $$f_{t,i}(i_1, \ldots, i_{m_1}, i_{m_1+1}, \ldots, i_{m-t}) = c_{t-1}((i_1, \ldots, i_{m_1}), i_{m_1+1}, \ldots, i_{m-t}, i)$$
      for any $(i_1, \ldots, i_{m-t}) \in H_1 \times \cdots \times H_{m-t}$.
      ii. P computes $\hat{f}_{t,i}(r_1, \ldots, r_{m-t})$ using Lemma 4.9, where $\hat{f}_{t,i} : (\mathbb{F}_{m-t})^{m-t} \to \mathbb{F}_{m-t}$ is the low degree extension of $f_{t,i}$, when viewed as a function $f_{t,i} : H_1 \times \cdots \times H_{m-t} \to \mathbb{F}_{m-t}$; Let $w_t(i) \in \mathbb{F}_{m-t}$ denote the output of the algorithm.
   (b) P sends $w_t \in (\mathbb{F}_{m-t})^{N_2'}$ to V.
   (c) For every $b \in B_{m-t}$, the verifier V checks that $w_t|_b \in (\mathbb{F}_1)^{N_2'}$ is a codeword of C_{N_2}.
   (d) Let $y_t \in (\mathbb{F}_{m-t})^{N_2}$ denote the systematic part of $w_t \in (\mathbb{F}_{m-t})^{N_2'}$. View y_t as a function $y_t : H_{m-(t-1)} \to \mathbb{F}_{m-(t-1)}$ (by identifying $[N_2]$ with $H_{m-(t-1)}$ and recalling that $\mathbb{F}_{m-t} \subseteq \mathbb{F}_{m-(t-1)}$), and let $\hat{y}_t : \mathbb{F}_{m-(t-1)} \to \mathbb{F}_{m-(t-1)}$ denote the low-degree extension of y_t. V checks that $\hat{y}_t(r_{m-(t-1)}) = v_{t-1}$.
   (e) V randomly chooses $\rho_{m_2-(t-1)} \in [N_2']$ and sends it to P.
   (f) Let $c_t \in C_{N_1} \otimes (C_{N_2})^{\otimes(m_2-t)}$ be defined as
   $$c_t(i, i_{m_1+1}, \ldots, i_{m-t}) = c_{t-1}(i, i_{m_1+1}, \ldots, i_{m-t}, \rho_{m_2-(t-1)})$$
   for any $i \in [N_1']$ and $i_{m_1+1}, \ldots, i_{m-t} \in [N_2']$, and let $v_t := w_t(\rho_{m_2-(t-1)}) \in \mathbb{F}_{m-t}$.
3. (a) P sends $w_{m_2+1} := c_{m_2} \in C_{N_1}$ to V.
   (b) The verifier V checks that w_{m_2+1} is a codeword of C_{N_1}.
   (c) Let $y_{m_2+1} \in (\mathbb{F}_1)^{N_1}$ denote the systematic part of $w_{m_2+1} \in (\mathbb{F}_1)^{N_1'}$. View y_{m_2+1} as a function $y_{m_2+1} : H_1 \times \cdots \times H_{m_1} \to \mathbb{F}_{m_1}$ (by identifying $[N_1]$ with $H_1 \times \cdots \times H_{m_1}$ and recalling that $\mathbb{F}_1 \subseteq \mathbb{F}_{m_1}$), and let $\hat{y}_{m_2+1} : \mathbb{F}_{m_1} \to \mathbb{F}_{m_1}$ denote the low-degree extension of y_{m_2+1}. V checks that $\hat{y}_{m_2+1}(r_1, \ldots, r_{m_1}) = v_{m_2}$.
   (d) V samples $\rho_0 \in [N_1']$ and outputs $\rho = (\rho_0, \rho_1, \ldots, \rho_{m_2})$ and $v_{m_2+1} := w_{m_2+1}(\rho_0)$.
4. V accepts if all of its checks passed, otherwise it rejects.

Figure 6: Fast Code Switching for LDE

Completeness. Completeness relies on the following claim.

Claim 4.12. *It holds that:*

1. $\hat{y}_1(r_m) = \hat{f}(r_1, \ldots, r_m)$.
2. *For any* $t \in \{1, 2, \ldots, m_2 - 1\}$, $\hat{y}_{t+1}(r_{m-t}) = w_t(\rho_{m_2-(t-1)})$.
3. $\hat{y}_{m_2+1}(r_1, \ldots, r_{m_1}) = w_{m_2}(\rho_1)$.

Proof. We prove each of the items separately.

Item 1: For any $(i_1, \ldots, i_m) \in H_1 \times \cdots \times H_m$, it holds that

$$\begin{aligned} f_{1,i_m}(i_1, \ldots, i_{m-1}) &= c_0((i_1, \ldots, i_{m_1}), i_{m_1+1}, \ldots, i_m) \\ &= M_f((i_1, \ldots, i_{m_1}), i_{m_1+1}, \ldots, i_m) \\ &= f(i_1, \ldots, i_m). \end{aligned}$$

Hence for any $i \in H_m$, the polynomials $\hat{f}_{1,i}(x_1, \ldots, x_{m-1})$ and $\hat{f}(x_1, \ldots, x_{m-1}, i)$ are two polynomials over $m-1$ variables $x_1, \ldots, x_{m-1}$, where each variable x_t has degree at most $|H_t| - 1$, that agree on all points of $H_1 \times \cdots \times H_{m-1}$, and so they are the same polynomial, and in particular,

$$w_1(i) = \hat{f}_{1,i}(r_1, \ldots, r_{m-1}) = \hat{f}(r_1, \ldots, r_{m-1}, i).$$

But this implies in turn that the polynomials $\hat{y}_1(x)$ and $\hat{f}(r_1, \ldots, r_{m-1}, x)$ are both univariate polynomials in the indeterminate x of degree $|H_m| - 1$ that agree on all points of H_m, so they are the same polynomial, and in particular $\hat{y}_1(r_m) = \hat{f}(r_1, \ldots, r_m)$.

Item 2: Next fix $t \in \{1, 2, \ldots, m_2 - 1\}$. For every $(i_1, \ldots, i_{m-t}) \in H_1 \times \cdots \times H_{m-t}$, it holds that

$$\begin{aligned} f_{t+1,i_{m-t}}\left(i_1, \ldots, i_{m-(t+1)}\right) &= c_t((i_1, \ldots, i_{m_1}), i_{m_1+1}, \ldots, i_{m-t}) \\ &= c_{t-1}\left((i_1, \ldots, i_{m_1}), i_{m_1+1}, \ldots, i_{m-t}, \rho_{m_2-(t-1)}\right) \\ &= f_{t,\rho_{m_2-(t-1)}}(i_1, \ldots, i_{m-t}). \end{aligned}$$

Hence for any $i \in H_{m-t}$, the polynomials $\hat{f}_{t+1,i}(x_1, \ldots, x_{m-(t+1)})$ and $\hat{f}_{t,\rho_{m_2-(t-1)}}(x_1, \ldots, x_{m-(t+1)}, i)$ are two polynomials over variables $x_1, \ldots, x_{m-(t+1)}$, where each variable x_t has degree at most $|H_t| - 1$, that agree on all points of $H_1 \times \cdots \times H_{m-(t+1)}$, and so they are the same polynomial, and in particular,

$$w_{t+1}(i) = \hat{f}_{t+1,i}(r_1, \ldots, r_{m-(t+1)}) = \hat{f}_{t,\rho_{m_2-(t-1)}}(r_1, \ldots, r_{m-(t+1)}, i).$$

But this implies in turn that the polynomials $\hat{y}_{t+1}(x)$ and $\hat{f}_{t,\rho_{m_2-(t-1)}}(r_1, \ldots, r_{m-(t+1)}, x)$ are both univariate polynomials of degree at most $|H_{m-t}| - 1$ that agree on all points of H_{m-t}, so they are the same polynomial, and in particular

$$\hat{y}_{t+1}(r_{m-t}) = \hat{f}_{t,\rho_{m_2-(t-1)}}(r_{m-t}) = w_t(\rho_{m_2-(t-1)}).$$

Item 3: For every $(i_1, \ldots, i_{m_1}) \in H_1 \times \cdots \times H_{m_1}$, it holds that

$$y_{m_2}(i_1, \ldots, i_{m_1}) = c_{m_2}(i_1, \ldots, i_{m_1}) = c_{m_2-1}((i_1, \ldots, i_{m_1}), \rho_1) = f_{m_2,\rho_1}(i_1, \ldots, i_{m_1}).$$

Hence the polynomials $\hat{y}_{m_2+1}(x_1,\ldots,x_{m_1})$ and $\hat{f}_{m_2,\rho_1}(x_1,\ldots,x_{m_1})$ are the same polynomial, and in particular

$$\hat{y}_{m_2+1}(r_1,\ldots,r_{m_1}) = \hat{f}_{m_2,\rho_1}(r_1,\ldots,r_{m_1}) = w_{m_2}(\rho_1).$$

□

Next assume that $\hat{f}(r) = v$, we shall show that in this case the verifier does not reject, and $c(\rho_0,\rho_1,\ldots,\rho_{m_2}) = v_{m_2+1}$.

For any $t \in [m_2]$ and $i \in [N_2']$, it is satisfied that $w_t(i) = \hat{f}_{t,i}(r_1,\ldots,r_{m-t})$. By linearity of the low-degree extension transformation, we get that w_t is a linear combination of the codewords in C_{N_2} over the field $\mathbb{F}_{m-t}$. Additionally, from the linearity of projection over base elements, we get that for every $b \in B_{m-t}$ it happens that $w_t|_b$ is a linear combination of the codewords of C_{N_2} and hence the verifier will not reject at Step 2c. We also clearly have that $w_{m_2+1} = c_{m_2}$ is a codeword of C_{N_1}, and so the verifier will not reject on Step 3b as well. Moreover, by assumption that $\hat{f}(r) = v$, and by the above Claim 4.12, the verifier does not reject on Steps 2d and 3c. Finally, note that $v_{m_2+1} = c_{m_2}(\rho_0) = c(\rho_0,\rho_1,\ldots,\rho_{m_2})$.

Soundness. Next assume that $\hat{f}(r) \neq v$. Fix a prover strategy P^*, and denote by $w_1^*, w_2^*, \ldots, w_{m_2}^*, w_{m_2+1}^*$ the messages that P^* sends in Steps 2b and 3a. For $t \in [m_2+1]$, let y_t^* denote the systematic part of w_t^*. Soundness relies on the following claim.

Claim 4.13. *If for some $t \in \{1,2,\ldots,m_2-1\}$, it holds that $w_t(\rho_{m_2-(t-1)}) \neq w_t^*(\rho_{m_2-(t-1)})$, and V does not reject on Steps 2c and 2d in iteration $t+1$, then with probability at least $1-\frac{\varepsilon}{m_2+1}$ over the choice of ρ_{m_2-t}, it holds that $w_{t+1}(\rho_{m_2-t}) \neq w_{t+1}^*(\rho_{m_2-t})$.*

Proof. Fix $t \in \{1,2,\ldots,m_2-1\}$. Since the verifier does not reject on Step 2d of iteration $t+1$, we get that $\hat{y}_{t+1}^*(r_{m-t}) = w_t^*(\rho_{m_2-(t-1)})$. On the other hand, by Claim 4.12, $\hat{y}_{t+1}(r_{m-t}) = w_t(\rho_{m_2-(t-1)})$. Given the assumption of the claim, this implies in turn that $\hat{y}_{t+1}(r_{m-t}) \neq \hat{y}_{t+1}^*(r_{m-t})$, and so $w_{t+1} \neq w_{t+1}^*$. Therefore there exists $b \in B_{m-t}$ such that $w_t|_b \neq w_t^*|_b$. Since V does not reject on Step 2c of iteration $t+1$, we further have that both $w_{t+1}|_b$ and $w_{t+1}^*|_b$ are codewords of C_{N_2}, and so they must differ on at least a $(1-\frac{\varepsilon}{m_2+1})$-fraction of the entries. Consequently, with probability at least $1-\frac{\varepsilon}{m_2+1}$ over the choice of $\rho_{m_2-t} \in [N_2']$, it holds that $w_{t+1}|_b(\rho_{m_2-t}) \neq w_{t+1}^*|_b(\rho_{m_2-t})$, and so also $w_{t+1}(\rho_{m_2-t}) \neq w_{t+1}^*(\rho_{m_2-t})$. □

Now, if the verifier rejects in any of the iterations then we are done. Hence we may assume that the verifier does not reject in any of the iterations, and we will show that in this case $c(\rho_0,\rho_1\ldots,\rho_{m_2}) \neq w_{m_2+1}^*(\rho_0)$ with probability at least $1-\varepsilon$ over the choice of $\rho_{m_2},\rho_{m_2-1},\ldots,\rho_0$.

By assumption that $\hat{f}(r) \neq v$ and by Claim 4.12, we have that $\hat{y_1}(r_m) \neq v_0$. On the other hand, by assumption that the verifier does not reject on Step 2d, we have that $\hat{y}_1^*(r_m) = v_0$. So we conclude that $w_1 \neq w_1^*$, and in particular $w_1|_b \neq w_1^*|_b$ for some $b \in B_{m-1}$. By assumption that the verifier does not reject on Step 2c, we have that $w_1|_b$ and $w_1^*|_b$ are both codewords of C_N, and so they differ by at least a $(1-\frac{\varepsilon}{m_2+1})$-fraction of the entries. Consequently, with probability at least $1-\frac{\varepsilon}{m_2+1}$ over the choice of ρ_{m_2}, we have that $w_1|_b(\rho_{m_2}) \neq w_1^*|_b(\rho_{m_2})$, and so $w_1(\rho_{m_2}) \neq w_1^*(\rho_{m_2})$. Applying Claim 4.13 iteratively, and using a union bound, we conclude that with probability at least $1-\frac{m_2}{m_2+1}\cdot\varepsilon$ over the choice of $\rho_{m_2},\rho_{m_2-1},\ldots,\rho_1$, it holds that $w_{m_2}(\rho_1) \neq w_{m_2}^*(\rho_1)$. Next assume that this latter event holds, i.e., that $w_{m_2}(\rho_1) \neq w_{m_2}^*(\rho_1)$.

Since the verifier does not reject on Step 3c, we get that $\hat{y}^*_{m_2+1}(r_1,\ldots,r_{m_1}) = w^*_{m_2}(\rho_1)$. On the other hand, by Claim 4.12, $\hat{y}_{m_2+1}(r_1,\ldots,r_{m_1}) = w_{m_2}(\rho_1)$. By assumption that $w_{m_2}(\rho_1) \neq w^*_{m_2}(\rho_1)$, this implies in turn that $\hat{y}_{m_2+1}(r_1,\ldots,r_{m_1}) \neq \hat{y}^*_{m_2+1}(r_1,\ldots,r_{m_1})$, and so $w_{m_2+1} \neq w^*_{m_2+1}$. Since V does not reject on Step 3b, we further have that both $w_{m_2+1} = c_{m_2}$ and $w^*_{m_2+1}$ are codewords of C_{N_1}, and so they must differ on at least a $(1-\frac{\varepsilon}{m_2+1})$-fraction of the entries. Consequently, with probability at least $1-\frac{\varepsilon}{m_2+1}$ over the choice of $\rho_0 \in [N'_1]$, it holds that $w_{m_2+1}(\rho_0) \neq w^*_{m_2+1}(\rho_0)$.

By a union bound, we conclude that with probability at least $1-\epsilon$ over the choice of $\rho_{m_2}, \rho_{m_2-1}, \ldots, \rho_0$, it holds that $w_{m_2+1}(\rho_0) \neq w^*_{m_2+1}(\rho_0)$. But since $w_{m_2+1}(\rho_0) = c_{m_2}(\rho_0) = c(\rho_0,\rho_1,\ldots,\rho_{m_2})$, this implies in turn that $w^*_{m_2+1}(\rho_0) \neq c(\rho_0,\rho_1,\ldots,\rho_\ell)$.

Number of rounds is clearly $m_2+1 = \lceil 1/\gamma \rceil + 1 = O(1)$. Next we analyze the communication and randomness complexity, and prover and verifier running time.

Communication and Randomness Complexity. For any $t \in [m_2]$, in round t the prover sends $w_t \in (\mathbb{F}_{m-t})^{N'_2}$, where each element of $\mathbb{F}_t$ can be represented using $O(\log(|\mathbb{F}_t|))$ bits, so the communication complexity is $N'_2 \cdot O(\log(\mathbb{F}_m)))$. In round m_2+1, the prover sends $w_{m_2+1} \in (\mathbb{F}_1)^{N'_1}$, so the communication complexity is $N' \cdot O(\log(\mathbb{F}_{n\cdot(\ell-t)})) \leq N'_1 \cdot O(\log(|\mathbb{F}_1|)) \leq N'_2 \cdot O(\log(\mathbb{F}_m)))$.

So the total communication complexity is at most

$$\begin{aligned}(m_2+1)\cdot N'_2 \cdot O(\log(|\mathbb{F}_m|)) &\leq (m_2+1)\cdot N_2 \cdot \mathrm{poly}(1/\varepsilon)\cdot O(\log(|\mathbb{F}_m|))\\ &\leq (m_2+1)\cdot n^{1/m_2}\cdot \mathrm{poly}(1/\varepsilon)\cdot O(\log(nm/\epsilon))\\ &\leq n^{\frac{\gamma}{2}}\cdot \mathrm{poly}(1/\varepsilon)\cdot O(\log(nm/\epsilon))\\ &\leq n^{\gamma}\cdot \mathrm{poly}(1/\varepsilon).\end{aligned}$$

Additionally, in every round $t \in [m_2]$, the verifier sends a single element in $[N'_2]$, and in round $[m_2+1]$, the verifier sends a single element in $[N'_2]$ and hence the total randomness complexity is at most

$$\begin{aligned}\log(N'_1) + m_2 \cdot \log(N'_2) &\leq \log(N_1\cdot \mathrm{poly}(1/\varepsilon)) + m_2 \cdot \log(N_2 \cdot \mathrm{poly}(1/\varepsilon))\\ &\leq \log(N_1 \cdot (N_2)^{m_2}) + O(\log(1/\varepsilon))\\ &\leq \log(n) + O(\log(1/\varepsilon)).\end{aligned}$$

Prover Complexity. For any $t \in [m_2]$ Step 2(a)ii of round t, the prover runs the algorithm from Lemma 4.9 for N'_2 times, on a function over the domain $H_1 \times \cdots \times H_{m-t}$. From the complexity analysis of the algorithm, this can be done via a circuit of size at most $\frac{n}{N_2}\cdot \mathrm{polylog}(1/\varepsilon)$, by Lemma 4.9. Thus, overall, in all rounds $t \in [m_2]$, the prover can be implemented as a Boolean circuit of size at most

$$\ell \cdot N'_2 \cdot \frac{n}{N_2}\cdot \mathrm{polylog}(1/\varepsilon) \leq \ell \cdot N_2 \cdot \mathrm{poly}(1/\varepsilon)\cdot \frac{n}{N_2}\cdot \mathrm{polylog}(1/\varepsilon) \leq n \cdot \mathrm{poly}(1/\varepsilon).$$

Additionally, in Step 3a, the prover sends a string $w_{m_2+1} \in (\mathbb{F}_1)^{N_1}$, which has length at most

$$N_1 \cdot O(\log(|\mathbb{F}_1|)) \leq N_1 \cdot \mathrm{poly}(1/\varepsilon) \leq (\ell_{m_1})^{m_1}\cdot \mathrm{poly}(1/\varepsilon) \leq 2^{\sqrt{\log n}\cdot O(\log^*(n))}\cdot \mathrm{poly}(1/\varepsilon) \leq n\cdot \mathrm{poly}(1/\varepsilon).$$

So in total, the prover can be implemented as a Boolean circuit of size at most $n\cdot \mathrm{poly}(1/\varepsilon)$.

Verifier Complexity. For every round $t \in [m_2]$, in Step 2c of round t, the verifier first projects the string $w_t \in (|\mathbb{F}_{m-t}|)^{N_2'}$ over all basis elements in B_{m-t}, which can be done in time $N_2' \cdot \text{polylog}(|\mathbb{F}_{m-t}|) \leq N_2 \cdot \text{poly}(1/\varepsilon) \cdot \text{polylog}(|\mathbb{F}_m|)$ by solving a system of linear equation. Then, the verifier checks that every projection is a codeword of C_{N_2} which, since C_{N_2} is a systematic code, can be done by re-encoding the word in overall time $N_2 \cdot \text{poly}(1/\varepsilon)$. In Step 2d of iteration t, the verifier computes the low-degree extension of $y_t \in (\mathbb{F}_{m-t})^{N_2}$ at a single point in $\mathbb{F}_{m-(t-1)}$, which can be done in time $\text{poly}(N_2, \log(|\mathbb{F}_{m-(t-1)}|)) \leq \text{poly}(N_2, \log(|\mathbb{F}_m|))$. So the total verifier complexity in the first m_2 rounds is at most

$$\text{poly}(N_2, 1/\varepsilon, \log(|\mathbb{F}_m|)) \leq \text{poly}(n^{1/m_2}, 1/\varepsilon, \log(nm/\varepsilon)) \leq n^{O(\gamma)} \cdot \text{poly}(1/\varepsilon).$$

In round m_2+1, in Step 3b, the verifier checks that w_{m_2+1} is a codeword of C_{N_1} which, since C_{N_1} is a systematic code, can be done by re-encoding the word in overall time $N_1 \cdot \text{poly}(1/\varepsilon)$. In Step 3c, the verifier computes the low-degree extension of $y_t \in (\mathbb{F}_1)^{N_1}$ at a single point in $H_1 \times \cdots \times H_m$, which can be done in time $\text{poly}(N_1, \log(|\mathbb{F}_{m_1}|)) \leq \text{poly}(N_1, \log(|\mathbb{F}_m|))$. So the total verifier complexity in round $m_2 + 1$ is at most

$$\begin{aligned}
\text{poly}(N_1, 1/\varepsilon, \log(|\mathbb{F}_m|)) &\leq \text{poly}((\ell_{m_1})^{m_1}, 1/\varepsilon, \log(nm/\varepsilon)) \\
&\leq \text{poly}\left(2^{\sqrt{\log n} \cdot O(\log^*(n))}, 1/\varepsilon, \log(nm/\varepsilon)\right) \\
&\leq n^{\gamma} \cdot \text{poly}(1/\varepsilon).
\end{aligned}$$

So overall the verifier running time is at most $n^{O(\gamma)} \cdot \text{poly}(1/\varepsilon)$.

Moreover part: The entry ρ clearly only depends on $\mathcal{V}$'s randomness. We also note that we could have replaced the code ensemble $\mathcal{C} = \{C_{N_1} \otimes (C_{N_2})^{\otimes m_2}\}_{n \in \mathbb{N}}$ with the code ensemble $\{(C_{(N_1)^{1/m_2}})^{\otimes m_2} \otimes (C_{N_2})^{\otimes m_2} = (C_{(N_1)^{1/m_2}} \otimes C_{N_2})^{\otimes m_2}\}_{n \in \mathbb{N}}$ in which each code is a tensor product of dimension $m_2 = \lceil 1/\gamma \rceil$, and this would not change the complexity analysis and any of the properties of the code and the protocol.

□

4.5 IOP for inner product check in log star rounds

Our Main Technical Theorem 4.1, which gives an *interactive proof* for inner product check in log star rounds, follows as a direct corollary of Lemma 4.5 from Section 4.2 and Lemma 4.11 from Section 4.4. Next we show how to deduce Corollary 4.2, which gives an *interactive oracle proof* (IOP) for inner product check in log star rounds, from Theorem 4.1 using proof composition. To this end, we use the following PCPP construction of Mie [Mie09].

Theorem 4.14. *Let $\mathcal{L}$ be a pair language decidable in time $T = T(m+n)$, where m, n are the explicit and implicit input lengths, respectively. Then for any $\alpha = \alpha(n) > 0$, there exists an $O(1/\alpha)$-query α-PCPP for $\mathcal{L}$ with constant soundness error, prover's running time $\text{poly}(m, n, T(m + \text{poly}(n)))$, and verifier's running time $\text{poly}(m, \log n, \log(T(m + \text{poly}(n))), 1/\alpha)$.*

Remark 4.15. *Mie [Mie09] gave a PCPP construction as above for any constant proximity parameter $\alpha > 0$. In Appendix A, we give a transformation which reduces the proximity parameter to any* subconstant *function $\alpha = \alpha(n)$, at the cost of increasing the query complexity by a factor of $O(1/\alpha)$, which proves the above theorem.*

Proof of Corollary 4.2. We start by setting some notation. Let $\mathcal{C}$ and $(\mathcal{P},\mathcal{V})$ be the code ensemble and the interactive protocol given by Theorem 4.1 for soundness error $\frac{\epsilon}{2}$ and a sufficiently small constant $\gamma' > 0$, to be determined later on. Let $m = O(\log^*(n))$ denote the round complexity of $(\mathcal{P},\mathcal{V})$. Without loss of generality, we may assume that all messages in the protocol $(\mathcal{P},\mathcal{V})$ have exactly the same length $\mathrm{cc} := n^{\gamma'} \cdot \mathrm{poly}(1/\varepsilon)$ (we can pad the messages to achieve this length without changing any of the properties of the protocol).

Let E be any polynomial-time encodable and decodable binary code ensemble of constant rate $\rho > 0$ and constant relative distance $\delta > 0$. For an element $a \in \mathbb{F}_1$, and an integer $t \geq 1$, let $a^{(t)} \in \mathbb{F}_1^t$ denote the string which consists of the concatenation of t copies of a. Let $\mathcal{L}_{\mathcal{V}} \subseteq \{0,1\}^*$ denote the pair language consisting of all strings of the form $(x_{\mathrm{exp}}, x_{\mathrm{imp}})$, where $x_{\mathrm{exp}} = (R, v)$ for $R \in \{0,1\}^*$ and $v \in \{0,1\}$, and $x_{\mathrm{imp}} = \left(E(z_1), \ldots, E(z_m), u_1^{(\mathrm{cc}/\rho)}, \ldots, u_d^{(\mathrm{cc}/\rho)}\right)$ for $z_i \in \{0,1\}^{\mathrm{cc}}$ and $u_j \in \mathbb{F}_1$, and where on input v, randomness string R, and prover's messages $z_1, \ldots, z_m$, $\mathcal{V}$ does not reject and outputs values $u_1, \ldots, u_d$. Let Π be the α-PCPP for $\mathcal{L}_{\mathcal{V}}$ given by Theorem 4.14 for $\alpha := \frac{\delta}{2(m+d\cdot\log(|\mathbb{F}_1|))}$, we may assume that Π has soundness error $\frac{\epsilon}{2}$, which can be achieved by repeating the verification step for $O(\log(1/\epsilon))$ times and accepting if and only if all invocations accept. Note that this increases the query complexity and verifier running time by a multiplicative factor of $O(\log(1/\epsilon))$.

The protocol: The protocol $(\mathcal{P}',\mathcal{V}')$ is obtained from $(\mathcal{P},\mathcal{V})$ via the following modifications. Let $z_1, \ldots, z_m$ denote the prover messages in $(\mathcal{P},\mathcal{V})$. For $i = 1, 2, \ldots, m$, $\mathcal{P}'$ sends the message $w_i = E(z_i) \in \{0,1\}^{\mathrm{cc}/\rho}$. Let R denote $\mathcal{V}$'s randomness string during the protocol, and let $i_1, \ldots, i_d \in [n']$ denote the indices that are output by $\mathcal{V}$ at the end of the protocol (whose location only depend on R). Then $\mathcal{P}'$ and $\mathcal{V}'$ run the PCPP Π on explicit input (R, v) and implicit input $w := \left(w_1, \ldots, w_m, (c_1(i_1))^{(\mathrm{cc}/\rho)}, \ldots, (c_d(i_d))^{(\mathrm{cc}/\rho)}\right)$, and $\mathcal{V}'$ accepts if and only if the PCPP verifier accepts (specifically, queries of the PCPP verifier $\mathcal{V}_{\mathsf{PCPP}}$ to the proof are answered using $\mathcal{V}'$'s PCPP oracle, and queries of $\mathcal{V}_{\mathsf{PCPP}}$ to any of the w_i's or c_j's are answered using $\mathcal{V}'$'s oracle w_i or c_j).

Complexity: It can be verified that the round complexity is $m + 1 = O(\log^*(n))$, the query complexity is $\log^*(n) \cdot O(\log(1/\epsilon))$, and the communication complexity is $n^{O(\gamma')} \cdot \mathrm{poly}(1/\varepsilon)$. The prover can be implemented as a Boolean circuit of size $(n + n^{O(\gamma')}) \cdot \mathrm{poly}(1/\varepsilon)$, and the verifier has running time $\mathrm{polylog}(n/\epsilon)$. Moreover, any code of $\mathcal{C}$ is a $\lceil 1/\gamma' \rceil$-dimensional tensor product. So all the claimed properties hold for a sufficiently small constant γ' (depending on γ). Completeness is also straightforward. Next we show soundness.

Soundness: Assume that $\sum_{i\in[n]} y_1(i) \cdots y_d(i) \neq v$, and fix a prover's strategy $(\mathcal{P}')^*$. Let $w_1^*, \ldots, w_m^*$ denote the messages of $(\mathcal{P}')^*$, and let R denote the randomness string of $\mathcal{V}'$. We shall show that with probability at least $1 - \frac{\varepsilon}{2}$ over the choice of R, w^* is α-far from $(\mathcal{L}_{\mathcal{V}})_{(R,v)}$, and so the PCPP verifier, and consequently also $\mathcal{V}'$, will reject with probability at least $1 - \frac{\epsilon}{2}$ (and hence the total rejection probability is at least $1 - \varepsilon$).

For $i \in [m]$, let $c_i^* = E(z_i^*) \in \{0,1\}^{\mathrm{cc}/\rho}$ be the codeword of E that is closest to w_i^*, and let $u_1^*, \ldots, u_d^* \in \mathbb{F}_1$ be the values output by $\mathcal{V}$ on input v, randomness string R, and prover's messages $z_1^*, \ldots, z_m^*$. Let $w^{**} = (c_1^*, \ldots, c_m^*, (u_1^*)^{(\mathrm{cc}/\rho)}, \ldots, (u_d^*)^{(\mathrm{cc}/\rho)})$. Note that by the soundness property of the protocol $(\mathcal{P},\mathcal{V})$, with probability at least $1 - \frac{\varepsilon}{2}$ over the choice of R, we have that $w^{**} \notin (\mathcal{L}_{\mathcal{V}})_{(R,v)}$. In what follows, assume that this event holds.

Suppose that $\tilde{w} = (\tilde{w}_1, \ldots, \tilde{w}_m, \tilde{v}_1, \ldots, \tilde{v}_d)$ is α-close to w^*, where $\tilde{w}_i \in \{0,1\}^{\mathrm{cc}/\rho}$ and $\tilde{v}_j \in \mathbb{F}_1^{(\mathrm{cc}/\rho)}$. We shall show that $\tilde{w} \notin (\mathcal{L}_{\mathcal{V}})_{(R,v)}$, and so w^* is at least α-far from $(\mathcal{L}_{\mathcal{V}})_{(R,v)}$. To see this, note that if there exists $i \in [m]$ so that $\tilde{w}_i$ is not a codeword of E, then clearly $\tilde{w} \notin (\mathcal{L}_{\mathcal{V}})_{(R,v)}$. Hence we may assume that all $\tilde{w}_i$ are codewords of E. Moreover, by assumption that $\tilde{w}$ is α-close to w^* for $\alpha < \frac{\delta}{2(m+d\cdot\log(|\mathbb{F}_1|))}$, we must have that $\mathrm{dist}(\tilde{w}_i, w_i^*) < \frac{\delta}{2}$ for any $i \in [m]$. But since E has relative distance at least δ, this implies in turn that $\tilde{w}_i = c_i^*$. Similarly, if there exists $j \in [d]$ so that $\tilde{v}_j$ is not the concatenation of cc/ρ identical copies of an element in $\mathbb{F}_1$, then clearly $\tilde{w} \notin (\mathcal{L}_{\mathcal{V}})_{(R,v)}$. Hence we may assume that for all $j \in [d]$, $\tilde{v}_j = (\tilde{u}_j)^{(\mathrm{cc}/\rho)}$ for some $\tilde{u}_j \in \mathbb{F}_1$. Moreover, by assumption that $\tilde{w}$ is α-close to w^* for $\alpha < \frac{1}{m+d\cdot\log(|\mathbb{F}_1|)}$, we must have that $\tilde{u}_j = u_j^*$ for all $j \in [d]$. So we conclude that $\tilde{w} = w^{**} \notin (\mathcal{L}_{\mathcal{V}})_{(R,v)}$.

□

5 From inner product check to IOPs for circuits

In this section we describe our IOP for circuit satisfiability using the inner product IOP of Theorem 4.1. Specifically, we target circuits that have some regularity properties: for example, a batch computation of the same function, or a sequentially composed function.

The construction is entirely standard, and follows the one in [HR22], which uses a (by now standard) arithmetization following Chiesa *et al.* [CHM+20]). For simplicity we target a constant soundness error, but remark that using the techniques from [HR22], the soundness error can be reduced to $2^{-\lambda}$ with only $\mathrm{polylog}(\lambda)$ multiplicative overhead.

It is convenient to describe the construction using the language of R1CS (for Rank 1 Constraint Satisfaction). Recall that R1CS is an NP-complete language, that is linear-time reducible from circuit SAT. In this problem we consider fixed matrices $A, B, C \in \mathbb{F}^{n\times n}$ and say that an input x is in the language if there exists w so that $(Az) \circ (Bz) = Cz$, where $z = x||w \in \mathbb{F}^S$ and $\circ$ denotes the pointwise product of the two vectors (aka Hadamard product). Following [BCR+19, HR22], we restrict our attention to matrices A, B and C which are tensor products of a $\mathrm{polylog}(n) \times \mathrm{polylog}(n)$ identity matrix and another another matrix of dimension $n/\mathrm{polylog}(n) \times n/\mathrm{polylog}(n)$.[9] Such R1CS are the instances that are derived by starting off from circuits that have a regular wiring pattern.

We construct an IOP for this R1CS problem as follows. Let $E : \{0,1\}^n \to \{0,1\}^{n'}$ be the sytematic linear-time encodable tensor code from Theorem 4.1 (using the terminology from that theorem, E is the tensor code $C^{\otimes t}$, but we use E to avoid the notation collision with the R1CS matrices). As E is a linear code, we use the notation Em to denote the encoding of the message m.

The prover computes $a = Az$, $b = Bz$ and $c = Cz$ and sends, as oracles, the codewords $\hat{z} = Ez$, $\hat{a} = Ea$, $\hat{b} = Eb$ and $\hat{c} = Ec$.

The prover needs to perform a few tests:

1. (Proximity Test:) Using the fact that E is a tensor code of dimension at least 3, by [Vid15] it is locally testable using $O(S^{2/3})$ queries for a codeword of size S. By composing with a PCPP of nearly linear length, the query complexity reduces to a constant and with a sub-linear length oracle.

   Actually, while we believe that the PCPP of [Mie09] can be implemented with a quasi-linear time prover, we are unaware of a formal proof of this fact. We sidestep this issue by making

[9] Actually [HR22] consider a richer class (called *tensor circuits*), but for simplicity we focus on the simplest case which covers, in particular, batch computation or repeated composition of a circuit.

E be a larger (but still constant) dimensional tensor (which we can, via Theorem 4.1). Via [Vid15] we can now reduce the query complexity to S^{ε}, for any desired constant $\varepsilon > 0$. Now, for any polynomial-time computable PCPP, we can choose ε to be sufficiently small so that we can afford the composition.

Once proximity of all oracles to corresponding codewords has been verified, the verifier can access the underlying codewords using the relaxed corrector for tensors of [GRR18], which are again composed with a PCPP to reduce the query complexity.

2. (Hadamard Test:) We want to check that $a \circ b \equiv c$. To do so, the verifier chooses a random vector $r \in \{0,1\}^n$ from a small-bias set. The test now reduces to checking that $\sum_{i\in[n]} r_i \cdot \hat{a}_i \cdot \hat{b}_i = \sum_{i\in[n]} r_i \cdot \hat{c}_i$. Each side of this equation can be directly verified using the inner product check of Theorem 4.1, in $O(\log^*(n))$ rounds. To run this check, the verifier needs to be able to emulate queries to the encoding Er of the small-bias sequence. This can be done using the fact that E is a (square) tensor code, and using a tensor based construction of a small-bias generator (see [RR25, Claim E.1.]).

3. (Lincheck Test:) We also need to check that $\hat{a}$, $\hat{b}$ and $\hat{c}$ are all consistent with $\hat{z}$. For simplicity we consider only checking consistency of $\hat{a}$ with $\hat{z}$ (and note that $\hat{b}$ and $\hat{c}$ can be handled similarly).

   Using the fact that E is a tensor code, and that A is the tensor product of a $\mathrm{polylog}(n) \times \mathrm{polylog}(n)$ identity matrix and another arbitrary matrix, following [HR22], we can reduce the problem by a $\mathrm{polylog}(n)$ factor and then solve it using PCPPs. This step also involves an initial preprocessing of the R1CS matrices. Using the fact that they are structured, this preprocessing step can also be done in $\mathrm{polylog}(n)$ time.

Overall the total number of rounds is $O(\log^*(S))$ as desired.

Finally, we remark that the IOP can be transformed into a succinct argument using the standard transformation [Kil92, BCS16] (see also the textbook [CY24]), while using the linear-size multi-selection circuit of [HR24] to efficiently compute the projections of the oracles to the queried locations.

Acknowledgments

We thank Oded Goldreich for inspiring this work by pointing out that the [RR25] protocol can be reduced to $O(\log\log(n))$ rounds, by composing it with a PCPP after $O(\log\log(n))$ rounds (along the lines of the sketch at the beginning of Section 1.2.2).

We also thank Giacomo Fenzi and Justin Thaler for helpful discussions.

References

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed-Solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 380–413. Springer, 2024. 7

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed-Solomon proximity testing with super-fast verification. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 214–243. Springer, 2025. 7

[ACY22] Gal Arnon, Alessandro Chiesa, and Eylon Yogev. A PCP theorem for interactive proofs and applications. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 64–94. Springer, 2022. 12

[ACY23] Gal Arnon, Alessandro Chiesa, and Eylon Yogev. IOPs with inverse polynomial soundness error. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 752–761. IEEE, 2023. 4

[AEL95] Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 512–519. IEEE Computer Society, 1995. 17

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017. 4

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 106–115. IEEE Computer Society, 2001. 3

[BBH+19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of Kilian-based SNARGs. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 522–551. Springer, 2019. 3

[BCF+25] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast SNARKs from interleaved RAA codes. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 123–152. Springer, 2025. 13

[BCFW25] Benedikt Bünz, Alessandro Chiesa, Giacomo Fenzi, and William Wang. Linear-time accumulation schemes. Cryptology ePrint Archive, Paper 2025/753, 2025. 13

[BCG+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 336–365. Springer, 2017. 4, 13

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 19–46. Springer, 2020. 5, 13

[BCGL22] Jonathan Bootle, Alessandro Chiesa, Ziyi Guan, and Siqi Liu. Linear-time probabilistic proofs with sublinear verification for algebraic automata over every field. *IACR Cryptol. ePrint Arch.*, page 1056, 2022. 13

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, pages 103–128, 2019. 47

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 31–60, 2016. 4, 14, 48

[BFK+24] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic SNARKs from expand-accumulate codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 276–307. Springer, 2024. 4

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991. 13

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput*, 36(4):889–974, 2006. 54, 55

[BMMS25] Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, and Matan Shtepel. FICS and FACS: Fast IOPPs and accumulation via code-switching. Cryptology ePrint Archive, Paper 2025/737, 2025. 13

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008. 11

[CCH+18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-Shamir from simpler assumptions. *IACR Cryptol. ePrint Arch.*, page 1004, 2018. 5

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1082–1090. ACM, 2019. 5

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020. 47

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. 48

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007. 11

[DP24] Benjamin E Diamond and Jim Posen. Polylogarithmic proofs for multilinears over binary towers. *Cryptology ePrint Archive*, 2024. 14

[DSW06] Irit Dinur, Madhu Sudan, and Avi Wigderson. Robust local testability of tensor products of LDPC codes. In *proceedings of the 9th International Workshop on Randomization and Computation (RANDOM)*, pages 304–315. Springer, 2006. 16

[DT24] Quang Dao and Justin Thaler. More optimizations to sum-check proving. *IACR Cryptol. ePrint Arch.*, page 1210, 2024. 9

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. 3

[GI05] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005. 17

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 102–113. IEEE Computer Society, 2003. 3

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023. 4, 13

[GRR18] Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 27:1–27:11, 2018. 6, 48

[Gru24] Angus Gruen. Some improvements for the PIOP for zerocheck. *IACR Cryptol. ePrint Arch.*, page 108, 2024. 13

[HR22] Justin Holmgren and Ron D. Rothblum. Faster sounder succinct arguments and IOPs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 474–503. Springer, 2022. 3, 4, 5, 13, 18, 47, 48

[HR24] Justin Holmgren and Ron Rothblum. Linear-size Boolean circuits for multiselection. In Rahul Santhanam, editor, *39th Computational Complexity Conference, CCC 2024, July 22-25, 2024, Ann Arbor, MI, USA*, volume 300 of *LIPIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. 5, 13, 48

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732, 1992. 48

[KRS25] Dmitry Khovratovich, Ron D. Rothblum, and Lev Soukhanov. How to prove false statements: Practical attacks on Fiat-Shamir. *IACR Cryptol. ePrint Arch.*, page 118, 2025. 3

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. 6, 19

[Mei13] Or Meir. IP = PSPACE using error-correcting codes. *SIAM J. Comput.*, 42(1):380–403, 2013. 10

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with O(1) queries. *Ann. Math. Artif. Intell*, 56(3-4):313–338, 2009. 11, 45, 47, 53, 54

[NST24] Vineet Nair, Ashish Sharma, and Bhargav Thankey. BrakingBase - a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme. *IACR Cryptol. ePrint Arch.*, page 1825, 2024. 13

[RR19] Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Electron. Colloquium Comput. Complex.*, TR19-127, 2019. 54

[RR24] Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *J. ACM*, 71(3):18, 2024. 4, 5, 10, 13, 25

[RR25] Noga Ron-Zewi and Ron Rothblum. Proving as fast as computing: Succinct arguments with constant prover overhead. *J. ACM*, 72(2), March 2025. 3, 4, 5, 6, 7, 10, 13, 14, 17, 48

[RRR21] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM J. Comput.*, 50(3), 2021. 4, 12, 14

[RW24] Noga Ron-Zewi and Mor Weiss. Zero-knowledge IOPs approaching witness length. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 105–137. Springer, 2024. 4, 13

[Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996. 6, 17

[Suc] Succinct. SP1 hardware acceleration. Accessed 6-17-2025. 3

[Sud01] Madhu Sudan. Algorithmic introduction to coding theory (lecture notes), 2001. 16

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. 9

[Vid15] Michael Viderman. A combination of testability and decodability by tensor products. *Random Structures and Algorithms*, 46(3):572–598, 2015. 6, 47, 48

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 223–237. IEEE Computer Society, 2013. 8, 9

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013. 17

[Wie88] Doug Wiedemann. An iterated quadratic extension of GF(2). *The Fibonacci Quarterly*, 26(4):290–295, 1988. 14

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022. 13

A PCPPs with improved proximity parameter dependence

In this section we prove Theorem 4.14. For this, we shall use the following PCPP due to [Mie09]. (The prover's running time is not explicitly stated in [Mie09], but can be deduced from the proof).

Theorem A.1 ([Mie09, Theorem 1]). *Let $\mathcal{L}$ be a pair language decidable in time $T = T(m+n)$, where m, n are the explicit and implicit input lengths, respectively. Then for any constant $\alpha > 0$, there exists a constant query α-PCPP for $\mathcal{L}$ with constant soundness error, prover's running time $\mathrm{poly}(m, n, T(m+n))$, and verifier's running time $\mathrm{poly}(m, \log n, \log(T(m+n)))$.*

Theorem A.1 gives a constant query PCPP with *constant* proximity parameter. Next we give a transformation that reduces the proximity parameter to any *subconstant* function $\alpha = \alpha(n)$, at the cost of increasing the query complexity by a factor of $O(1/\alpha)$. A similar transformation was given in [RR19, Lemma 8.6], albeit with a larger overhead of $\mathrm{poly}(1/\alpha)$ in the query complexity. The transformation relies on the following notion of a *relaxed locally decodable code* [BGH+06].

Definition A.2 (Relaxed Locally Decodable Codes (RLDCs)). *Let $C : \Sigma^k \to \Sigma^n$ be an error correcting code. We say that C is a q-query relaxed locally decodable code (RLDC) from α-fraction of errors if there exists a randomized oracle machine M, called the local decoder, which gets as input oracle access to $w \in \Sigma^n$ and explicit access to an index $i \in [k]$, makes at most q queries to the oracle and satisfies the following two conditions.*

1. *If $w = C(m)$, then $M^w(i) = m_i$ with probability 1.*
2. *If w is α-close to some codeword $c = C(m) \in C$, then with probability at least $\frac{1}{2}$ it holds that $M^w(i)$ either outputs m_i or a special abort symbol $\perp$.*

Lemma A.3. *Let $\mathcal{L}$ be a pair language, and let $\alpha = \alpha(n) > 0$. Suppose that the following exist.*

- *A relaxed locally decodable code $\mathcal{C} = \{C_n : \{0,1\}^n \to \{0,1\}^{n'}\}_{n \in \mathbb{N}}$ with relative distance δ, decoding radius $\frac{\delta}{2}$, and query complexity $q_{rlcc}(n)$.*
- *A PCPP for $\mathcal{L}' := \{(x, C(w)) : (x, w) \in \mathcal{L}\}$ with communication complexity $\mathrm{cc}(n)$, query complexity $q(n)$, proximity parameter $\frac{\delta}{2}$, and soundness error $\frac{1}{2}$, where n is the length of the implicit input w.*

Then, there exists a PCPP for $\mathcal{L}$ with communication complexity $n' + \mathrm{cc}(n)$, query complexity $q(n) + q_{rlcc}(n) \cdot O(1/\alpha)$, proximity parameter α, and soundness error $\frac{1}{2}$, where n is the length of the implicit input w.

Moreover,

- *If the verifier in the PCPP for $\mathcal{L}'$ has running time $T(n)$, and the relaxed local decoder for $\mathcal{C}$ has running time time $T_{rlcc}(n)$, then the verifier in the resulting PCPP has running time $T(n) + T_{rlcc}(n) \cdot O(1/\alpha)$.*
- *If the prover in the PCPP for $\mathcal{L}'$ has running time $T(n)$, and $\mathcal{C}$ has encoding time $T_{enc}(n)$, then the prover in the resulting PCPP has running time $T(n) + T_{enc}(n)$.*

Proof. The PCPP Π for $\mathcal{L}$ consists of the string $C(w)$, followed by the PCPP Π' for $\mathcal{L}'$ with respect to the explicit input x and implicit input $C(w)$. Let z denote the first part of the proof Π that is allegedly equal to $C(w)$.

In the query phase, the verifier first runs the check of Π', with respect to the explicit input x and the implicit input z. If Π' rejects, then the verifier rejects. Otherwise, the verifier repeats the following procedure $O(1/\alpha)$ times: the verifier picks a uniform random $i \in [n]$, and applies the relaxed local decoder $\mathcal{C}$ on input coordinate i with oracle access to z. If in any of the invocations

the output of the relaxed local decoder is different than w_i (which it obtains by making a query to the implicit input w) then the verifier rejects. Otherwise, the verifier accepts.

It can be verified that the communication complexity, query complexity, and verifier and prover running times are all as claimed. Completeness is also straightforward. Next we show soundness.

Suppose that w is α-far from $\mathcal{L}_x$. If z is $\frac{\delta}{2}$-far from any codeword of C then Π' rejects with probability at least $\frac{1}{2}$. Hence we may assume that z is $\frac{\delta}{2}$-close to some codeword $C(w')$. Next assume that $w' \notin \mathcal{L}_x$. Then, since the code C has relative distance at least δ, we have that $C(w')$ is δ-far from $\mathcal{L}'_x$. By the triangle inequality, this implies in turn that z is $\frac{\delta}{2}$-far from $\mathcal{L}'_x$, and so Π' will once more reject with probability at least $\frac{1}{2}$. Thus, we may also assume that $C(w') \in \mathcal{L}'_x$.

By definition, if $C(w') \in \mathcal{L}'_x$ then $w' \in \mathcal{L}_x$. But as we have assumed that w is α-far from $\mathcal{L}_x$, the above implies in turn that $\mathrm{dist}(w, w') \geq \alpha$. Thus, with probability at least α, the verifier will pick $i \in [n]$ on which w and w' differ, and moreover, the relaxed local decoder will output w'_i or $\perp$ with probability at least $\frac{1}{2}$. As both events are independent, we conclude that the verifier rejects with probability at least $\frac{\alpha}{2}$ on each of the invocations. Finally, repeating this procedure for $O(1/\alpha)$ times gives rejection probability at least $\frac{1}{2}$. □

Theorem 4.14 follows by instantiating Lemma A.3 with an explicit rLDC code with constant relative distance, constant query complexity, polynomial length, and poly-logarithmic running time (e.g., the rLDC of [BGH+06] suffices[10]).

[10] [BGH+06] do not explicitly state the running time of their rLDCs. However, they construct rLDCs out of PCPPs in a black-box way, where the running time of the rLDC is the same as the running time of the PCPP verifier. The running time of the rLDC can be made polylogarithmic by instantiating it with a PCPP of constant query complexity, polynomial length, and logarithmic verifier running time (e.g., the PCPP given by Theorem A.1).

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