LiLAC: Linear Prover, Logarithmic Verifier and Field-agnostic Multilinear Polynomial Commitment Scheme

Kyeongtae Lee[1], Seongho Park[1], Byeongjun Jang[2], Jihye Kim[2,3], and Hyunok Oh[1,3]

[1] Information Systems, Hanyang University, Seoul, Republic of Korea
{rsias9049,seonghopark,hoh}@hanyang.ac.kr
[2] Electrical Engineering, Kookmin University, Seoul, Republic of Korea
{sunjbs,jihyek}@kookmin.ac.kr
[3] Zkrypto Inc.

Abstract. In this paper, we propose LiLAC, a novel field-agnostic, transparent multilinear polynomial commitment scheme (MLPCS) designed to address key challenges in polynomial commitment systems. For a polynomial with N coefficients, LiLAC achieves $\mathcal{O}(N)$ prover time, $\mathcal{O}(\log N)$ verifier time, and $\mathcal{O}(\log N)$ proof size, overcoming the limitations of $\mathcal{O}(\log^2 N)$ verification time and proof size without any increase in other costs. This is achieved through an optimized polynomial commitment strategy and the recursive application of the tensor IOPP, making LiLAC both theoretically optimal and practical for large-scale applications. Furthermore, LiLAC offers post-quantum security, providing robust protection against future quantum computing threats.

We propose two constructions of LiLAC: a field-agnostic LiLAC and a field-specific LiLAC. Each construction demonstrates superior performance compared to the state-of-the-art techniques in their respective categories of MLPCS. First, the field-agnostic LiLAC is compared against Brakedown (CRYPTO 2023), which is based on a tensor IOP and satisfies field-agnosticity. In experiments conducted over a 128-bit field with a coefficient size of 2^{30}, the field-agnostic LiLAC achieves a proof size that is 3.7× smaller and a verification speed that is 2.2× faster, while maintaining a similar proof generation time compared to Brakedown. Furthermore, the field-specific LiLAC is evaluated against WHIR (ePrint 2024/1586), which is based on an FRI. With a 128-bit field and a coefficient size of 2^{30}, the field-specific LiLAC achieves a proof generation speed that is 2.8× faster, a proof size that is 27% smaller, and a verification speed that is 14% faster compared to WHIR.

Table of Contents

1 Introduction

A Succinct Non-interactive Argument of Knowledge (SNARK) is a cryptographic primitive that enables a prover to convince a verifier that it possesses a valid witness for an NP statement with sublinear verification costs [PHGR16, WTS+18, GWC19, BFS20, Set20, CBBZ23], making SNARKs crucial in numerous real-world applications where computations must be verified by resource-constrained parties.

A common approach to constructing a SNARK combines the Polynomial Interactive Oracle Proof (PIOP) [BFS20] with a Polynomial Commitment Scheme (PCS) [KZG10, PST13, ZGK+17, AHIV17, BSCR+19, ZXZS20, BFS20, Set20]. In a polynomial IOP, the prover provides the verifier with truth tables of large polynomials. A PCS allows the prover to commit to a polynomial f using a short commitment and later prove evaluation claims like $f(x) = y$. By combining these, an efficient proof system is achieved: the prover commits to the polynomials using PCS, providing values and proofs of correctness to the verifier.

The concept of polynomial commitment schemes was first introduced by Kate et al. [KZG10], who constructed a univariate polynomial commitment scheme using bilinear groups. This foundational work led to several extensions into the multivariate setting, allowing for the handling of more complex data structures and computations [PST13, ZGK+18]. As a result, numerous multilinear polynomial commitment schemes with efficient provers have since been developed, including Hyrax [WTS+18], Brakedown [GLS+23], Orion [XZS22], and Basefold [ZCF23].

Since the efficiency of a SNARK is largely determined by the performance of its underlying PCS, optimizing PCS has become a central research focus. Important efficiency metrics include proving time, verification time, and proof size, with recent studies exploring trade-offs among these factors. Schemes with a (universal) trusted setup [KZG10, ZXZS20] achieve constant proof size and verification time but sacrifice plausible post-quantum security. In contrast, transparent PCS constructions [BBB+18, WTS+18, Lee21, BSBHR18] avoid the need for trusted setups or complex multi-party computation, though they typically result in larger proof sizes and verification times. Field-agnostic schemes [GLS+23, ZCF23] further enhance versatility by efficiently supporting diverse field sizes, from binary fields in hardware verification to large prime fields in cryptographic applications.

In this work, we focus on designing efficient and practical Multilinear Polynomial Commitment Schemes (MLPCS). Our goals are to create transparent MLPCS that do not require a trusted setup, to minimize proving and verifying times as well as proof size, and to ensure compatibility with arbitrary (sufficiently large) fields, i.e., to be field-agnostic. Recent literature [ZCF23] shows that transparent, field-agnostic schemes generally achieve linear-time proving, polylogarithmic-time verification, and polylogarithmic proof sizes. Although proving time has been significantly optimized to approach near-optimal linear complexity, the barriers of polylogarithmic complexity in verification costs and communication overhead remain. This emphasizes the need for a transparent MLPCS

with smaller proofs and faster verification to support a wider range of applications. We aim to design a transparent, field-agnostic MLPCS that achieves asymptotically *logarithmic*-time verification and *logarithmic*-sized proofs, while maintaining linear-time proving. We utilize the hash function to provide plausible post-quantum security.

Table 1: Comparison to Existing Code-based Polynomial Commitment Schemes. In the comparison table, N represents the size of the coefficients of a polynomial. In this table, Prover complexity and Verifier complexity indicate the number of field multiplications, while Proof size represents the number of field elements. We denote ℓ as the testing parameter.

Protocol	Prover complexity	Proof size	Verifier complexity	Field-agnostic
FRI [BSBHR18]	$\mathcal{O}(N \log N)$	$\mathcal{O}(\ell \log^2 N)$	$\mathcal{O}(\ell \log^2 N)$	No
Brakedown [GLS+23]	$\mathcal{O}(N)$	$\mathcal{O}(\ell\sqrt{N})$	$\mathcal{O}(\ell\sqrt{N})$	Yes
Orion [XZS22]	$\mathcal{O}(N)$	$\mathcal{O}(\ell \log^2 N)$	$\mathcal{O}(\ell \log^2 N)$	No
BaseFold [ZCF23]	$\mathcal{O}(N \log N)$	$\mathcal{O}(\ell \log^2 N)$	$\mathcal{O}(\ell \log^2 N)$	Yes
STIR [ACFY24a]	$\mathcal{O}(N \log N)$	$\mathcal{O}(\ell \log\log N \cdot \log N)$	$\mathcal{O}(\ell \log\log N \cdot \log N + \ell^2 \log N)$	No
WHIR [ACFY24b]	$\mathcal{O}(N \log N)$	$\mathcal{O}(\ell \log\log N \cdot \log N)$	$\mathcal{O}(\ell \log\log N \cdot \log N + \log N)$	No
Ours	$\mathcal{O}(N)$	$\mathcal{O}(\ell \log N)$	$\mathcal{O}(\ell \log N)$	Yes

Transparent, field-agnostic PCSs can be built using Interactive Oracle Proofs of Proximity (IOPP) [BSCG+16,RRR16], a specialized proof system that verifies whether a committed vector over a field $\mathbb{F}$ is close to a codeword in a linear error-correcting code (ECC) $C \subseteq \mathbb{F}$. There are two main types of IOPP-based PCS constructions: FRI (Fast Reed-Solomon Interactive Proof of Proximity) for Reed-Solomon codes and tensor code-based IOPPs.

FRI. FRI uses Reed-Solomon codes, which have a high relative distance, thereby requiring fewer testing parameters and resulting in a smaller proof size. However, Reed-Solomon codes rely on the Fast Fourier Transform (FFT), which imposes a computational complexity of at least $\mathcal{O}(N \log N)$ for the prover. Moreover, FFT is highly dependent on the choice of the field, making Reed-Solomon codes field-specific. BaseFold [ZCF23] extends the FRI protocol to construct an efficient commitment scheme specifically for multilinear polynomials, and it verifies polynomial evaluations by integrating with the classical sum-check protocol. BaseFold achieves a prover complexity of $\mathcal{O}(N \log N)$, while the verifier complexity remains $\mathcal{O}(\log^2 N)$. Notably, unlike other FRI-based approaches, BaseFold is field-agnostic, which significantly enhances its utility for SNARK applications by employing foldable linear codes. STIR [ACFY24a] improves upon the structure of FRI by introducing a folding technique to reduce the code rate at each round, decreasing the verifier's query complexity to $\mathcal{O}(\log N + \lambda \cdot \log\log N)$. Through this approach, STIR achieves smaller proof sizes and faster verification times, while also maintaining $\mathcal{O}(N \log N)$ prover time. STIR enhances security and efficiency

compared to FRI by requiring fewer queries for verification. WHIR [ACFY24b] introduces constrained Reed–Solomon Codes to construct an efficient IOPP, supporting both multivariate and univariate polynomials. While maintaining the same query complexity as STIR, WHIR significantly improves verification speed. Specifically, the verification time of STIR is $\mathcal{O}(\lambda \log\log N + \lambda^2 \log N)$, whereas WHIR achieves $\mathcal{O}(\lambda \log\log N + \log N)$, enabling faster verification. However, despite its speed advantage over STIR, WHIR still does not achieve field-agnosticism.

Tensor IOPP. Approaches based on tensor code-based IOPP reduce the proving time to $\mathcal{O}(N)$ by using linear-time encodable code. However, compared to FRI-based approaches, this approach generally leads to higher verification costs, even if they exhibit the same time complexity. This is because linear-time encodable codes pose a lower relative distance, requiring more testing parameters for proximity and consistency checks. Bootle et al. [BCG20] proposed a new type of IOP using tensor codes, achieving linear prover time and quasi-linear verifier time. This allows for efficient proofs for R1CS problems with millions of constraints, designed to work in various fields. This approach significantly improved the efficiency of polynomial verification and demonstrated the potential for linear-time proof generation. Brakedown [GLS+23] is based on the polynomial commitment scheme proposed in [BCG20]. Brakedown's key contribution is achieving field-agnosticism while offering a linear prover time, using linear-time encodable codes. Its proof size and verifier time to test the proximity and consistency of the code remain $\mathcal{O}(\sqrt{N})$. Orion [XZS22] improves the verifier cost and proof size to $\mathcal{O}(\log^2 N)$, keeping the linear-time prover through the use of proof recursion, which encodes the verifier computation into a general-purpose SNARK circuit which is not field-agnostic.

1.1 Our Contributions

Optimized Multilinear Polynomial Commitment Scheme. We introduce a novel, field-agnostic, and transparent MLPCS that combines tensor IOPP with the sum-check protocol, referred to as LiLAC. Our approach achieves $\mathcal{O}(N)$ proving time, $\mathcal{O}(\log N)$ verification time, and $\mathcal{O}(\log N)$ proof size. To the best of our knowledge, LiLAC is the first scheme to achieve both logarithmic verification time and proof size, improving upon prior polylogarithmic bounds, for a field-agnostic and transparent MLPCS with linear-time proving.

Recursive Reduction with Improved Efficiency. We propose a recursive reduction framework that transforms large polynomial evaluations into smaller, manageable ones, enabling efficient final verification. By merging proximity and consistency tests into a single operation during the reduction steps, our approach minimizes computational overhead and enhances the verification process. The minimized proof size and verification time of $\mathcal{O}(\log N)$ increases practicality in resource-constrained environments.

Flexible Code and PCS Selection. LiLAC supports flexible selection of error-correcting codes and polynomial commitment schemes for final verification, allowing performance optimization based on application needs. For example, combining Spielman codes with Brakedown or BaseFold PCS provides linear proving time and field-agnosticism, while Reed-Solomon codes with FRI-based PCS offer smaller verification times and proof sizes. This adaptability ensures LiLAC can tailor its code and PCS choices to achieve optimal performance across diverse use cases.

Concrete Performance Improvements. Section 4 presents experimental results for LiLAC, including the field-agnostic variants LiLAC(Brakedown) and LiLAC(Basefold), which use Brakedown and Basefold respectively, and the field-specific variant LiLAC(Shockwave), which utilizes Shockwave.

In a 128-bit field with 128-bit security, LiLAC(Brakedown) achieves proof generation times comparable to Brakedown for input sizes of 2^{30} while reducing proof sizes by 3.7× and doubling the verification speed. Under the same conditions, LiLAC(Basefold) offers the smallest proof size among existing field-agnostic MLPCSs at 30MB and the fastest verification time at 0.19 seconds, while maintaining proving times similar to other MLPCSs.

Furthermore, in a 128-bit field with 100-bit security, comparisons against the field-specific MLPCSs Orion and WHIR show that, for input sizes of 2^{30}, LiLAC(Shockwave) achieves proving times that are 2.8× faster than WHIR and 5× faster than Orion. LiLAC(Shockwave) also delivers proof sizes that are 27% smaller and verification speeds that are 14% faster than WHIR. Compared to Orion, LiLAC(Shockwave) achieves proof sizes that are 52× smaller and verification speeds that are 87× faster.

The comprehensive comparisons in Table 3 and Table 4 demonstrate LiLAC's balanced improvements in proving time, proof size, and verification speed across various input sizes. In particular, our scheme is significantly more efficient for large input sizes, highlighting its critical role in efficiently handling large datasets and high-degree polynomials.

1.2 Technical Overview

We first explore the method of constructing a multilinear polynomial commitment scheme (MLPCS) using tensor IOPP (Interactive Oracle Proof of Proximity). The core idea of tensor IOPP is to represent a multilinear polynomial with N coefficients as a matrix, then apply a linear error correction code to each row (or column) to create a matrix containing codewords. This process is referred to as *encoding*. On the other hand, we can multiply the matrix representing N coefficients by an appropriate vector to create a vector expressed as a linear combination, known as *folding*. As shown in Figure 1, we apply the fold and encode operations in reverse on the encoded matrix and folded vector, respectively. Since both fold and encode are commutative linear operations, the results from both sides should be identical, yielding $\text{fold}(\text{encode}(\mathbf{C})) = \text{encode}(\text{fold}(\mathbf{C}))$, where $\mathbf{C}$ represents the matrix of N coefficients.

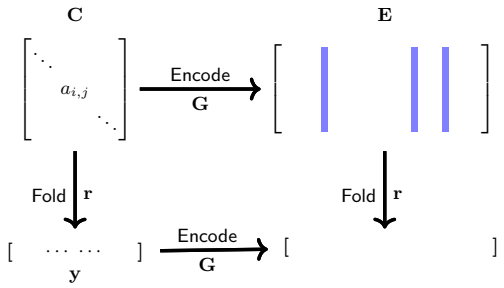


Fig. 1: Proximity and consistency tests in tensor IOPP

For each element in the vectors derived from the left and right sides, tensor IOPP guarantees, with high probability, that the two vectors are identical if a sufficient number of random point elements match through proximity testing of the codeword. To ensure that the column vectors are part of encode($\mathbf{C}$), we first commit to encode($\mathbf{C}$) by constructing a Merkle tree from its column vectors. We then combine these Merkle trees into another Merkle tree along the row direction, using the resulting Merkle root as the commitment value for encode($\mathbf{C}$). In existing tensor IOPP approaches, vector equality is verified either directly by the verifier [BCG20,GLS+23] or through the use of SNARKs [XZS22].

In our approach, we use a sum-check protocol to verify random position values in the equation fold(encode($\mathbf{C}$)) = encode(fold($\mathbf{C}$)). The fold operation on the codeword matrix from encode($\mathbf{C}$) is represented by multiplying it with a vector, while encoding the vector fold($\mathbf{C}$) involves applying the encode matrix. Denoting the codeword matrix as $\mathbf{E}$, the encode matrix as $\mathbf{G}$, the vector fold($\mathbf{C}$) as $\mathbf{y}$, and the vector for fold($\mathbf{E}$) as $\mathbf{r}$, we have:

$$\sum_i \tilde{G}(j,i)\tilde{y}(i) = \sum_i \tilde{E}(j,i)\tilde{r}(i),$$

for all rows (or columns) where $\tilde{X}(j,i)$ represents a multilinear polynomial to represent $\mathbf{X}$.

When sum-check is applied to verify arbitrary multilinear polynomial evaluations, the problem reduces to checking the polynomial evaluation at random values. However, applying this directly to the above equations would require evaluating the polynomials $\tilde{G}$ and $\tilde{E}$, which contain more than N elements, negating any efficiency gains from using the sum-check protocol.

We observe that tensor IOPP requires verification of only a limited number of random positions, defined by the security parameter λ as a testing parameter ℓ[4]. To address this, we construct reduced matrices $\mathbf{G}'$ and $\mathbf{E}'$ that contain elements for only the selected positions in $\mathbf{G}$ and $\mathbf{E}$ and perform sum-check on this re-

[4] Here, ℓ is the testing parameter for tensor IOPP; in Brakedown [GLS+23], it is referred to as the number of column openings, and in Orion [XZS22], it is called the opening.

duced form. To facilitate this, our proposed scheme first constructs Merkle trees for both $\mathbf{G}$ and $\mathbf{E}$ to create commitments. Specifically, as shown in Figure 2, we extract Merkle tree intermediate nodes representing the ℓ selected rows (or columns) from the committed $\mathbf{G}$ and $\mathbf{E}$ matrices, creating a new commitment for the next round of recursion.

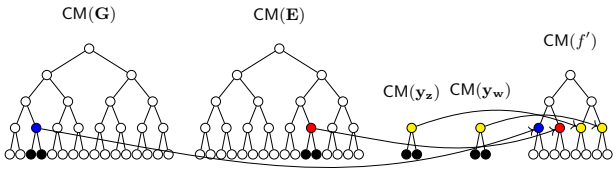


Fig. 2: Committing process for the next commitment

The newly constructed $\mathbf{G}'$ and $\mathbf{E}'$ matrices are reduced to $\ell\sqrt{N}$ in size compared to the original $\mathbf{G}$ and $\mathbf{E}$. This reduction is possible because if $\mathbf{C}$, the matrix representing a polynomial with N coefficients, is square, each row and column has $\sqrt{N}$ elements. Selecting ℓ such rows (or columns) forms the reduced matrices $\mathbf{G}'$ and $\mathbf{E}'$.

By combining all MLPCSs for $\mathbf{G}'$, $\mathbf{E}'$, $\mathbf{y}$, and $\mathbf{r}$ into a single MLPCS, we evaluate this combined MLPCS to confirm the evaluations of $\mathbf{G}'$, $\mathbf{E}'$, $\mathbf{y}$, and $\mathbf{r}$ match correctly. This process invokes our MLPCS once more to evaluate the newly created MLPCS, which now has $2\ell\sqrt{N}+2\sqrt{N}$ coefficients. More precisely, the number of coefficients N' for the next round can be expressed as follows:

$$N' = \psi(2\ell\sqrt{N} + 2\sqrt{N}),$$

where $\psi(x)$ is the smallest power of 2 that is not less than x, represented as 2^{2k} for some integer k. This ensures that $\sqrt{N'}$ for the next round N' has a power-of-2 form. This process repeats recursively until the coefficient count cannot be reduced further, at which point we use an existing MLPCS to complete the proof.

In each round, we need to evaluate the coefficient matrix $\mathbf{C}_i$ with respect to the evaluation value vector $\mathbf{z}$. When independently performing MLPCS for each round, the prover applies encoding to the given coefficient matrix $\mathbf{C}_i$, and the verifier, after receiving the commitment $\mathsf{cm}_{\mathbf{E}_i}$ of $\mathbf{E}_i(=\text{encode}(\mathbf{C}_i))$, checks whether $\mathbf{E}_i$ has been properly encoded. To do this, the verifier selects a random vector $\mathbf{w}$ and performs both the proximity test to verify $\mathbf{E}_i$ and the consistency test to ensure correctness with respect to the given vector $\mathbf{z}$. If the value $\mathbf{z}$ is randomly selected after $\mathbf{E}_i$ has been committed, the proximity test and consistency test can be combined into a single test [DP24]. Therefore, in the proposed scheme, we first perform the commitment for encode(fold($\mathbf{C}_{i+1}$)) for the next round before determining $\mathbf{z}$ in the sum-check process. This order allows us to apply an optimization that merges $\mathbf{z}$ and $\mathbf{w}$ into a single vector.

For a polynomial with N_i coefficients, the prover needs $\mathcal{O}(N_i)$[5] to compute fold and $\mathcal{O}(N_i)$ to compute encode (assuming the encode function is linear time). The prover sends Merkle roots for the ℓ positions received from the verifier by extracting them from $\mathbf{G}$ and $\mathbf{E}$. Additionally, membership proofs (with $\mathcal{O}(\log N_i)$ hash values) are included to verify that each Merkle root belongs to $\mathbf{G}$ or $\mathbf{E}$. In the sum-check phase, the prover and verifier perform $\log N_i$ interactions, with the verifier receiving $\mathcal{O}(\log N_i)$ values from the prover. Thus, the total proving time for the prover is $\mathcal{O}(N_i)$, while the total verifying time for the verifier is $\mathcal{O}(\log N_i)$, with a proof size of $\mathcal{O}(\log N_i)$.

With each recursive MLPCS reduction, the coefficient count decreases to $\sqrt{N_i}$, and $N_i = \sqrt{N_{i-1}} = N^{\frac{1}{2^i}}$. Therefore, the total proving time is:

$$\sum_{i=0}^{\infty} \mathcal{O}(N^{\frac{1}{2^i}}) = \mathcal{O}(N),$$

and the verifying time and proof size are both:

$$\sum_{i=0}^{\infty} \mathcal{O}(\log N^{\frac{1}{2^i}}) = \mathcal{O}(\log N).$$

Note that the overhead from the final MLPCS evaluation is constant, as it involves a polynomial with a fixed number of coefficients.

1.3 Outline

The rest of this paper is organized as follows: Section 2 introduces cryptographic backgrounds. Section 3 presents our LiLAC, a field-agnostic and transparent MLPCS with a linear time prover and a logarithmic verifier. We also implement LiLAC and offer its experimental results and comparison with other PCS in Section 4. Section 5 concludes the paper.

2 Preliminaries

Let $[N]$ denote the set of integers $\{1, \ldots, N\}$. Given a set S, we use the notation $a \leftarrow\!\$\, S$ to indicate that an element a is sampled uniformly at random from S. We denote by λ the security parameter. Let $\mathbb{F}[x_1, \ldots, x_n]$ represent the polynomial ring over the field $\mathbb{F}$, where $\mathbb{F}$ is a field of prime order p. Vectors and matrices are represented in bold font. For a vector $\mathbf{v} \in \mathbb{F}^n$, the components are denoted by $v_1, \ldots, v_n$. Similarly, for a matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$, the (i, j)-th element of $\mathbf{M}$ is denoted by $\mathbf{M}[i, j]$. The notation $\mathbf{M}[i, :]$ refers to the i-th row of $\mathbf{M}$, while $\mathbf{M}[:, j]$ denotes the j-th column of $\mathbf{M}$, where $i \in [m]$ and $j \in [n]$.

[5] We analyze the complexity while ignoring the constant ℓ in the testing parameter.

2.1 Multilinear Extension

Suppose $f : \{0,1\}^d \to \mathbb{F}$ is a function that maps d-bit elements into an element of $\mathbb{F}$. A polynomial extension of f is a low-degree d-variate polynomial $\tilde{f}(\cdot)$ such that $\tilde{f}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^d$.

A multilinear extension (or MLE) is a low-degree polynomial extension where the extension is a multilinear polynomial. Given a function $F : \{0,1\}^d \to \mathbb{F}$, the multilinear extension of $F(\cdot)$ is the unique multilinear polynomial $\tilde{F} : \mathbb{F}^d \to \mathbb{F}$. It can be computed as follows.

$$\begin{aligned}\tilde{F}(\mathbf{x}) &= \sum_{e \in \{0,1\}^d} F(e) \cdot \prod_{i=1}^{d} (x_i \cdot e_i + (1 - x_i) \cdot (1 - e_i)) \\ &= \sum_{e \in \{0,1\}^m} F(e) \cdot \widetilde{\mathrm{eq}}(\mathbf{x}, \mathbf{e})\end{aligned}$$

where $\widetilde{eq}(\mathbf{x}, \mathbf{e})$ is the MLE of the following function:

$$\mathrm{eq}(\mathbf{x}, \mathbf{e}) = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{e} \\ 0 & \text{otherwise} \end{cases}$$

2.2 Polynomial Commitment Schemes

A Polynomial Commitment Scheme (PCS) is a tuple of four PPT protocols $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ defined as follows:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d)$: Setup takes as input the security parameter λ and the number of variables in a multilinear polynomial d and outputs the public parameters pp.
- $C \leftarrow \mathsf{Commit}(\mathsf{pp}, f)$: Commit takes as input pp and a multilinear polynomial $f \in \mathbb{F}[X_0, \ldots, X_{d-1}]$ and outputs a commitment C.
- $1/0 \leftarrow \mathsf{Open}(\mathsf{pp}, f, C)$: Open takes as input pp, f, and C, and outputs 1 if C is a commitment to f and 0 otherwise.
- $1/0 \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \mathbf{z}, \sigma, d, f)$: Eval is an interactive protocol between a probabilistic prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The public inputs are pp, C, a evaluation point $\mathbf{z} \in \mathbb{F}^d$ and a value $\sigma \in \mathbb{F}$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $f(\mathbf{z}) = \sigma$, and $\mathcal{V}$ outputs 1 if it is convinced and 0 otherwise.

A PCS $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ must additionally satisfy binding, completeness, and knowledge-soundness.

Definition 1 (Binding). *For any PPT adversary $\mathcal{A}$, and $d \geq 1$ the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{l} \mathit{Open}(\mathit{pp}, C, f_0, o_0) = 1 \\ \wedge\ \mathit{Open}(\mathit{pp}, C, f_1, o_1) = 1 \\ \wedge\ f_0 \neq f_1 \end{array} \;\middle|\; \begin{array}{l} \mathit{pp} \leftarrow \mathit{Setup}(1^\lambda, N), \\ (C, f_0, f_1, o_0, o_1) \leftarrow \mathcal{A}(\mathit{pp}, d) \end{array}\right]$$

Definition 2 (Completeness). *For any multilinear polynomial f the following probability is 1*

$$\Pr\left[\begin{array}{l|l} \mathsf{Eval}(\mathsf{pp}, C, \mathbf{z}, \sigma; f) = 1 & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N) \\ \wedge\ f(\mathbf{z}) = \sigma & C \leftarrow \mathsf{Commit}(\mathsf{pp}, f) \\ & 1 \leftarrow \mathsf{Open}(\mathsf{pp}, C, f, o) \end{array}\right]$$

Definition 3 (Knowledge-Soundness). *Eval is an argument of knowledge for the following relation given $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, d)$:*

$$\mathcal{R} = \left\{ (C, \mathbf{v}, \sigma; f) \;\middle|\; f \in \mathbb{F}[X_0, \ldots, X_{d-1}] \ \wedge\ \mathbf{v} \in \mathbb{F}^d \ \wedge\ \sigma \in \mathbb{F} \ \wedge\ f(\mathbf{v}) = \sigma \right\}$$

2.3 Merkle Tree

A Merkle tree is a data structure that allows for committing to $\ell = 2^{\mathsf{dep}}$ messages with a single hash value h, such that revealing any single message bit requires $\mathsf{dep} + 1$ hash values.

Definition 4 (Merkle Tree). *A Merkle tree is represented by a binary tree of depth dep, where ℓ message elements $m_1, m_2, \cdots, m_\ell$ are assigned to the leaves of the tree. The values assigned to the internal nodes are computed by hashing the values of their two child nodes. To reveal m_i, one must provide m_i together with the values on the path from m_i to the root. The algorithms are as follows:*

- $h \leftarrow \mathsf{Merkle.CM}(m_1, \cdots, m_\ell)$*: Computes the root hash h of the Merkle tree.*
- $(m_i, \pi_i) \leftarrow \mathsf{Merkle.Open}(\mathbf{m}, i)$*: Given the messages and an index i, outputs the message m_i and the proof π_i.*
- $1/0 \leftarrow \mathsf{Merkle.Verify}(\pi_i, m_i, h)$*: Given the proof π_i, the message m_i, and the root Hash h, outputs 1 if the proof is valid and 0 otherwise.*

2.4 Linear Code

Definition 5 (Linear Code [XZS22]). *A $[n, k, d]$ linear code $\mathcal{C}$ is a collection of functions $c : [n] \to \mathbb{F}$ that form an $\mathbb{F}$-linear subspace in $\mathbb{F}^n$ where n is the codeword length, k is the message length, and d is the minimum distance of the code. The rate ρ of the code is defined as the ratio of the message length to the codeword length, $\rho = \frac{k}{n}$, and the relative distance δ is defined as $\frac{d}{n}$, representing the minimum fractional distance between distinct codewords.*

There exist maps that can expand or reduce the size of a given tensor using linear combinations.

Definition 6 (Fold). *The folding of a function $f : [n_1] \times [n_2] \times \cdots \times [n_t] \to \mathbb{F}$ by a linear combination $\alpha : [n_1] \to \mathbb{F}$ is the function $\mathsf{Fold}(f; \alpha) : [n_2] \times \cdots \times [n_t] \to \mathbb{F}$ defined as follows:*

$$\mathsf{Fold}(f; \alpha) := \sum_{i=1}^{n_1} \alpha(i) \cdot f(i, \cdot, \cdots, \cdot).$$

Messages from $\mathbb{F}^k$ are encoded via an injective linear function $\mathsf{Enc} : \mathbb{F}^k \to \mathcal{C}$. Specifically, a message $m \in \mathbb{F}^k$ is transformed to Gm, where $G \in \mathbb{F}^{n\times k}$ is the generator matrix of the code. This can be generalized as follows:

Definition 7 (Encode). *Let $G : [n] \times [k] \to \mathbb{F}$ be the generator matrix of a linear code $\mathcal{C}$ in $\mathbb{F}^n$. Let $f : [k] \times [n_1] \times \cdots \times [n_t] \to \mathbb{F}$ be a function. The encoding of f is the function $\mathsf{Enc}(f) : [n] \times [n_1] \times \cdots \times [n_t] \to \mathbb{F}$ defined as follows:*

$$\mathsf{Enc}(f)(j, \cdot, \cdots, \cdot) := \sum_{i=1}^{k} f(i, \cdot, \cdots, \cdot) G(i, j).$$

Definition 8. *Let $f : [k]^t \to \mathbb{F}$ be a message function with inputs indexed by $(i_1, i_2, \cdots, i_t)$. The $\mathcal{C}^{\otimes t}$-encoding of f is the function $\mathsf{Enc}_{1\cdots,t}(f) : [n]^t \to \mathbb{F}$, indexed by $(j_1, j_2, \cdots, j_t)$, defined as follows:*

$$\begin{aligned}&\mathsf{Enc}_{1\cdots,t}(f)(j_1, \cdots, j_t)\\ &\quad := \sum_{i_1=1}^{k} \cdots \sum_{i_t=1}^{k} f(i_1, \cdots, i_t) G(i_1, j_1) \cdots G(i_t, j_t).\end{aligned}$$

Proposition 1 (Lemma 7.8 [BCG20]) *Consider the following functions:*

$$\begin{aligned} f_{r-1} &: [\ell] \times [k]^{t-r+1} \to \mathbb{F},\\ c_{r-1} &: [\ell] \times [n]^{t-r+1} \to \mathbb{F},\\ f_r &: [k]^{t-r+1} \to \mathbb{F},\\ c_r &: [k] \times [n]^{t-r} \to \mathbb{F},\\ q &: [\ell] \to \mathbb{F}\end{aligned}$$

satisfying the following conditions:

$$\begin{aligned} c_{r-1} = \mathsf{Enc}_{r,\cdots,t}(f_{r-1}), \quad f_r = \mathsf{Fold}_{r-1}(f_{r-1}; q),\\ c_r = \mathsf{Enc}_{r+1,\cdots,t}(f_r).\end{aligned}$$

Then, it holds that $\mathsf{Fold}_{r-1}(c_{r-1}; q) = \mathsf{Enc}_r(c_r)$. In other words, the following diagram commutes:

$$\begin{array}{ccccc} \mathbb{F}^{\ell k^{t-r+1}} & & \xrightarrow{\quad\mathsf{Enc}_{r,\cdots,t}\quad} & & \mathbb{F}^{\ell n^{t-r+1}} \\ \Big\downarrow{\scriptstyle \mathsf{Fold}_{r-1}} & & \circlearrowleft & & \Big\downarrow{\scriptstyle \mathsf{Fold}_{r-1}} \\ \mathbb{F}^{k^{t-r+1}} & \xrightarrow{\mathsf{Enc}_{r+1,\cdots,t}} & \mathbb{F}^{kn^{t-r}} & \xrightarrow{\mathsf{Enc}_r} & \mathbb{F}^{n^{t-r+1}} \end{array}$$

2.5 Sum-check Protocol

The sum-check protocol [LFKN92] allows a verifier to offload the computation of the following sum to an untrusted prover:

$$\sum_{x_n \in \{0,1\}} \cdots \sum_{x_2 \in \{0,1\}} \sum_{x_1 \in \{0,1\}} f(x_1, x_2, \ldots, x_n),$$

where f is an n-variate polynomial over a finite field $\mathbb{F}$. For the sake of brevity and clarity in this paper, we will denote the above sum more concisely as:

$$\sum_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}), \text{ where } \mathbf{x} = (x_1, x_2, \ldots, x_n).$$

3 Multilinear Polynomial Commitment Schemes with a Logarithmic Verifier

In this section, we present LiLAC, a field-agnostic and transparent multilinear polynomial commitment scheme with a linear prover and logarithmic verifier. Our MLPCS, LiLAC, recursively applies the reduction process denoted as LiLAC-Reduce to simplify the multilinear polynomial and finally applies MLPCS at the end.

Let $\mathbf{C}$ be a $\sqrt{N}\times\sqrt{N}$-matrix. Let $f_N^{\mathbf{C}}$ be a multilinear polynomial in $\mathbb{F}[x_1, \ldots, x_{\log N}]$ with coefficients matrix $\mathbf{C}$. Given a vector $\mathbf{z} \in \mathbb{F}^{\log N}$ and a scalar $\sigma \in \mathbb{F}$ and commitment to $f_N^{\mathbf{C}}$, the prover must demonstrate to the verifier that $\sigma = f_N^{\mathbf{C}}(\mathbf{z})$. We utilize a Merkle tree (see Section 2.3), which has a hierarchical structure, and denote this commitment as Merkle.CM, and Table 2 provides an overview of the notation and variables used in our proposed algorithm.

Table 2: Notations and variables

N	The size of the coefficients such that $N = 2^r = k^2$
$\mathbf{C}$	The coefficients represented as a matrix in $\mathbb{F}^N$
N_i	The i-th size of the coefficients such that $N_i = k_i^2$
$\mathbf{C}_i$	The i-th coefficient matrix in $\mathbb{F}^{N_i}$
$\mathcal{C}_i$	The $[n_i, k_i, d_i]$-linear (and systematic) code, where n_i is $\rho^{-1}k_i = 2k_i$, with relative distance $\delta = \frac{d_i}{n_i}$
$\mathbf{G}_i$	The i-th encoding matrix $\left[\frac{\mathbf{I}}{\mathbf{P}_i}\right]$ in $\mathbb{F}^{n_i\times k_i}$ where $\mathbf{I}$ is the identity matrix of size $k_i \times k_i$ and $\mathbf{P}_i$ is a $(k_i \times k_i)$-matrix
Enc_i	The i-th encoding function
Fold	The folding map
$\psi(x)$	It returns the smallest value v, where $v \geq x$, and $\sqrt{v}$ is a power-of-2 form
ℓ	The testing parameter for proximity and consistency checks

Our LiLAC consists of four components: Setup, Commit, Open and Eval (see Figure 3). In Setup, the number of rounds rn for LiLAC-Reduce is set, and the encoding matrices are generated and committed, outputting the public parameter pp. In Commit, the prover generates the commitment for the coefficient matrix

$\mathbf{C}$. In Open, the verifier validates the opening of the commitment. In Eval, the prover claims the evaluation of f at $\mathbf{z}$ as σ.

The prover encodes the coefficient matrix $\mathbf{C}$, computes the commitment for a submatrix $\mathbf{D}$, and sends it to the verifier. Initially, we assume that all encodings utilize systematic codes, which are error-correcting codes that preserve the original data unchanged while appending extra bits for error detection or correction. For the proximity test, the verifier provides a random vector $\mathbf{w}$ to the prover. Using the parameters $(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}}, \mathsf{cm}_{\mathbf{D}}, \mathbf{z}, \mathbf{w}, \sigma)$, the prover and verifier invoke LiLAC-Reduce. The primary goal of LiLAC-Reduce is to reduce the size N and the coefficient matrix $\mathbf{C}$ (corresponding to the multilinear polynomial $f_N^{\mathbf{C}}$) to $N' = \psi(2(\ell+1)\sqrt{N})$. This process generates a smaller coefficient matrix $\mathbf{C}' \in \mathbb{F}^{\sqrt{N'} \times \sqrt{N'}}$, resulting in a simplified multilinear polynomial $f_{N'}^{\mathbf{C}'}$.

By evaluating the polynomial at the reduced size N' rather than the original size N, the protocol significantly reduces both computational cost and proof size. The reduction process, which assumes systematic codes, is described in Section 3.2. After completing all reduction steps recursively, the construction of LiLAC is finalized by applying the multilinear polynomial commitment scheme (MLPCS), as described in Section 3.1.

In cases where non-systematic codes are employed, additional verification is required to confirm that the codeword $\mathbf{E}$ is correctly derived from $\mathbf{C}$. This verification is performed using the sum-check protocol, as detailed in Section 3.3, which extends the scheme to support non-systematic codes.

3.1 LiLAC-MLPCS

Through the recursive structure of LiLAC-Reduce, invoked by our proposed multilinear polynomial commitment scheme LiLAC, substantial reductions in both verification costs and proof size are achieved, bringing them to logarithmic scale.

Setup for LiLAC. During Setup, the necessary information for the encoding function used in each reduction is computed. This includes the message length, codeword length, and details of the encoding matrix. The data for all rounds is stored, and the public parameters are generated, including the commitments to the encoding matrices.

Let rn denote the number of recursion rounds in LiLAC-Reduce, which depends on both N and ℓ. For each round in LiLAC-Reduce, the number of coefficients N is reduced to $\psi(2(\ell+1)\sqrt{N})$. The recursion continues until the size of N stabilizes relative to $\psi(2(\ell+1)\sqrt{N})$, ensuring that further reductions are no longer necessary. At this point, rn is determined. For each $i \in [\mathsf{rn}]$, there exists an encoding matrix $\mathbf{G}_i \in \mathbb{F}^{k_i \times n_i}$ for the encoding function $\mathsf{Enc}_i : k_i \to n_i$. The party computes the Merkle commitment for each matrix $\mathbf{G}_i$, resulting in commitments denoted as $\mathsf{cm}_{\mathbf{G}_i}$. Let pp be the set consisting of $\{\mathsf{cm}_{\mathbf{G}_i}\}$, and $\{\mathbf{G}_i\}$ for all $1 \le i \le \mathsf{rn}$. The party then returns pp.

Commit for LiLAC. In Commit, the prover computes the Merkle commitment for the coefficient matrix $\mathbf{C}$.

$\mathsf{Setup}(1^\lambda, N)$

$\mathcal{C} \leftarrow \{\text{Collection of Linear Codes}\}$
$\ell \leftarrow \text{constructparamater}(\lambda, \mathcal{C})$
$\mathsf{rn} \leftarrow \min\{i \mid N_i \leq N_{i+1}\}$
  where $N_1 = N$ and $N_{i+1} = \psi(2(\ell+1)\sqrt{N_i})$
for $1 \leq i \leq \mathsf{rn}$,
  $\mathbf{G}_i \leftarrow \text{constructEncodeMatrix}(\mathcal{C}, N_i)$
  $\mathsf{cm}_{\mathbf{G}_i} \leftarrow \mathsf{Merkle.CM}(\mathbf{G}_i)$
$\mathsf{pp} = (\{\mathsf{cm}_{\mathbf{G}_i}, \mathbf{G}_i\}_{1 \leq i \leq \mathsf{rn}})$
return pp

$\mathsf{Commit}(\mathsf{pp}, f_N^{\mathbf{C}})$

$\mathbf{C} \leftarrow f_N^{\mathbf{C}}$
$\mathsf{cm}_{\mathbf{C}} \leftarrow \mathsf{Merkle.CM}(\mathbf{C})$
return $\mathsf{cm}_{\mathbf{C}}$

$\mathsf{Open}(\mathsf{pp}, f_N^{\mathbf{C}}, \mathsf{cm}_{\mathbf{C}})$

$\mathbf{C} \leftarrow f_N^{\mathbf{C}}$
$\mathcal{V}$ accepts if $\mathsf{cm}_{\mathbf{C}} = \mathsf{Merkle.CM}(\mathbf{C})$

$\mathsf{Eval}(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}}, \mathbf{z}, \sigma; f_N^{\mathbf{C}})$

$\mathcal{P}$ claims $\sigma = f_N^{\mathbf{C}}(\mathbf{z})$ to $\mathcal{V}$
$\mathcal{P}$ computes $\mathbf{E} = \mathsf{Enc}(\mathbf{C})$
$\mathcal{P}$ sets $\mathbf{D} = (\mathbf{E}[i,j])_{k+1 \leq i \leq n}^{1 \leq j \leq k}$
$\mathcal{P}$ computes $\mathsf{cm}_{\mathbf{D}} = \mathsf{Merkle.CM}(\mathbf{D})$ and sends it to $\mathcal{V}$
$\mathcal{V}$ samples $\mathbf{w} \leftarrow\$ \mathbb{F}^{\log N}$ and sends it to $\mathcal{P}$
$\mathcal{P}$ and $\mathcal{V}$ call the $\mathsf{LiLAC\text{-}Reduce}(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}}, \mathsf{cm}_{\mathbf{D}}, \mathbf{z}, \mathbf{w}, \sigma; \mathbf{C}, \mathbf{E})$
$\mathcal{V}$ accepts if $1 \leftarrow \mathsf{LiLAC\text{-}Reduce}$

Fig. 3: LiLAC

Open for LiLAC. In Open, the verifier verifies that $\mathsf{cm}_{\mathbf{C}}$ is a valid commitment to f.

Evaluation for LiLAC. In Eval, the prover and verifier recursively execute LiLAC-Reduce to verify whether the evaluation of the polynomial $f_N^{\mathbf{C}}$ at $\mathbf{z}$ equals σ. The prover begins by applying the encoding function Enc to encode each column of the matrix $\mathbf{C} \in \mathbb{F}^{k \times k}$, resulting in the codeword $\mathbf{E} = \mathsf{Enc}(\mathbf{C}) \in \mathbb{F}^{n \times k}$. In case that a systematic code is used, the codeword $\mathbf{E}$ can be expressed as

$\left[\frac{\mathbf{C}}{\mathbf{D}}\right]$ where $\mathbf{D} \in \mathbb{F}^{(n-k)\times k}$ is a submatrix, allowing for efficient separation of the original data $\mathbf{C}$ and the additional parity bits $\mathbf{D}$. The prover commits to $\mathbf{D}$ using Merkle commitments and sends this commitment, denoted as $\mathsf{cm}_{\mathbf{D}}$, to the verifier. The verifier then samples a random point $\mathbf{w} \leftarrow\!\!\$\ \mathbb{F}^{\log N}$ and sends it to the prover. Next, the prover and verifier execute LiLAC-Reduce with the input $(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}}, \mathsf{cm}_{\mathbf{D}}, \mathbf{z}, \mathbf{w}, \sigma; \mathbf{C}, \mathbf{E})$. The verifier accepts the result if LiLAC-Reduce outputs 1.

3.2 LiLAC-Reduce

In this section, we describe the process LiLAC-Reduce that simplifies a multilinear polynomial by reducing the number of its coefficients, producing a smaller multilinear polynomial.

The LiLAC-Reduce process consists primarily of two stages: the tensor IOPP process and the sum-check process. In the tensor IOPP process, the prover starts with a matrix $\mathbf{C}$, constructed from the coefficients of a multilinear polynomial f, and an encoded matrix $\mathbf{E}$, which is generated by applying an encoding function to each column of $\mathbf{C}$. Both the prover and the verifier have Merkle tree commitments for $\mathbf{C}$ and $\mathbf{E}$, denoted by $\mathsf{cm}_{\mathbf{C}}$ and $\mathsf{cm}_{\mathbf{E}}$, respectively. The process takes as inputs the vector $\mathbf{z}$, which contains values substituted for the variables of f, and the evaluation result $\sigma = f(\mathbf{z})$. Additionally, a random vector $\mathbf{w}$, selected by the verifier for the proximity test, is provided as an input. The steps of the LiLAC-Reduce process are as follows:

- **Folding**: Using the input vector $\mathbf{z}$ and the verifier-provided vector $\mathbf{w}$, the prover folds the coefficient matrix $\mathbf{C}$, computing $\mathbf{y}_{\mathbf{z}}$ and $\mathbf{y}_{\mathbf{w}}$.
- **Membership Proof**: The verifier selects positions to check, and a membership proof is provided for the corresponding intermediate Merkle tree roots representing rows at those positions.
- **Commitment for the Next Round**: A new coefficient matrix $\mathbf{C}'$ for the next round is generated, and its encoding and corresponding commitment $\mathsf{cm}_{\mathbf{E}'}$ are computed.
- **Multilinear Extension and Sum-Check**: The sum-check protocol is used to verify the proximity and consistency tests required in the tensor IOPP, as well as to validate the tensor product calculations. The sum-check process transitions to the multilinear polynomial evaluation of a random vector $\mathbf{z}'$. Since the coefficient matrix $\mathbf{C}'$, intended for the next round, is already encoded and committed as $\mathsf{cm}_{\mathbf{E}'}$, the same random vector $\mathbf{z}'$ can also be used for the proximity test, eliminating the need to select a separate random vector $\mathbf{w}'$.
- **Invoke for the Next Round**: This step is performed to initiate the MLPCS evaluation for the subsequent round, aiming to prove the evaluation result $\sigma' = f(\mathbf{z}')$.

LiLAC-Reduce$(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}}, \mathsf{cm}_{\mathbf{D}}, \mathbf{z}, \mathbf{w}, \sigma; \mathbf{C}, \mathbf{E})$

1 : **if** $N \leq N' = \psi(2(\ell+1)\sqrt{N})$,

2 : $\mathcal{P}$ and $\mathcal{V}$ execute a MLPCS as a black box to verify that $f_N^{\mathbf{C}}(\mathbf{z}) = \sigma$.

3 : $\mathcal{V}$ outputs 1 if the check passes, 0 otherwise

4 : **else**

5 : $\mathcal{P}$ computes $\mathbf{r}_{\mathbf{z}} = \otimes_{i=\log k+1}^{r}(1-z_i, z_i)$, $\mathbf{y}_{\mathbf{z}} = \mathsf{Fold}(\mathbf{C}; \mathbf{r}_{\mathbf{z}})$

6 : $\mathcal{P}$ computes $\mathbf{r}_{\mathbf{w}} = \otimes_{i=\log k+1}^{r}(1-w_i, w_i)$, $\mathbf{y}_{\mathbf{w}} = \mathsf{Fold}(\mathbf{C}; \mathbf{r}_{\mathbf{w}})$

7 : $\mathcal{P}$ computes $\mathsf{cm}_{\mathbf{y}_{\mathbf{z}}} = \mathsf{Merkle.CM}(\mathbf{y}_{\mathbf{z}})$, $\mathsf{cm}_{\mathbf{y}_{\mathbf{w}}} = \mathsf{Merkle.CM}(\mathbf{y}_{\mathbf{w}})$ and sends it to $\mathcal{V}$

8 : $\mathcal{V}$ computes $\mathsf{cm}_{\mathbf{E}} = \mathsf{Merkle.CM}(\mathsf{cm}_{\mathbf{C}}, \mathsf{cm}_{\mathbf{D}})$

9 : $\mathcal{V}$ samples $I = \{l_1, \cdots, l_\ell\} \subset [n]$ and sends it to $\mathcal{P}$

10 : $\mathcal{P}$ computes $\mathsf{cm}_{\mathbf{E}[l_t:]} = \mathsf{Merkle.CM}(\mathbf{E}[l_t, :])$ and $\mathsf{cm}_{\mathbf{G}_{[l_t:]}} = \mathsf{Merkle.CM}(\mathbf{G}[l_t, :])$ for all $l_t \in I$

11 : $\mathcal{P}$ sends $\mathsf{cm}_{\mathbf{E}[l_t:]}$, $\mathsf{cm}_{\mathbf{G}[l_t:]}$, and co-paths to $\mathcal{V}$ for all $t \in [\ell]$

12 : $\mathcal{V}$ checks $\mathsf{cm}_{\mathbf{E}[l_t:]} \in \mathsf{cm}_{\mathbf{E}}$ and $\mathsf{cm}_{\mathbf{G}[l_t:]} \in \mathsf{cm}_{\mathbf{G}}$ for all $t \in [\ell]$

13 : $\mathcal{P}$ sets $\mathbf{C}' = (\mathbf{G}[l_1, :], \ldots, \mathbf{G}[l_\ell :], \underbrace{0, \cdots, 0}_{\frac{N'}{2}-k\ell-k}, \mathbf{y}_{\mathbf{z}}, \mathbf{E}[l_1, :], \ldots, \mathbf{E}[l_\ell :], \underbrace{0, \cdots, 0}_{\frac{N'}{2}-k\ell-k}, \mathbf{y}_{\mathbf{w}}) \in \mathbb{F}^{N'}$

14 : $\mathcal{P}$ computes $\mathbf{E}' = \mathsf{Enc}'(\mathbf{C}')$ and sets $\mathbf{D}' = \mathbf{E}'[i,j]_{k'+1 \leq i \leq n'}^{1 \leq j \leq k'}$

15 : $\mathcal{P}$ computes $\mathsf{cm}_{\mathbf{D}'} = \mathsf{Merkle.CM}(\mathbf{D}')$ and sends it to $\mathcal{V}$

16 : $\mathcal{P}$ and $\mathcal{V}$ compute $\widetilde{l}_{\mathbf{z}}(\mathbf{x})$, $\widetilde{r}_{\mathbf{z}}(\mathbf{x})$, and $\widetilde{r}_{\mathbf{w}}(\mathbf{x})$

17 : $\mathcal{P}$ computes $\widetilde{y}_{\mathbf{z}}(\mathbf{x})$, $\widetilde{y}_{\mathbf{w}}(\mathbf{x})$, $\widetilde{G}(\mathbf{y}, \mathbf{x})$ and $\widetilde{E}(\mathbf{y}, \mathbf{x})$

18 : $\mathcal{P}$ computes $S_{\mathbf{z}}(\mathbf{y}, \mathbf{x}) = \widetilde{G}(\mathbf{y}, \mathbf{x})\widetilde{y}_{\mathbf{z}}(\mathbf{x}) - \widetilde{E}(\mathbf{y}, \mathbf{x})\widetilde{r}_{\mathbf{z}}(\mathbf{x})$ and $T(\mathbf{x}) = \widetilde{l}_{\mathbf{z}}(\mathbf{x})\widetilde{y}_{\mathbf{z}}(\mathbf{x})$

19 : $\mathcal{P}$ computes $S_{\mathbf{w}}(\mathbf{y}, \mathbf{x}) = \widetilde{G}(\mathbf{y}, \mathbf{x})\widetilde{y}_{\mathbf{w}}(\mathbf{x}) - \widetilde{E}(\mathbf{y}, \mathbf{x})\widetilde{r}_{\mathbf{w}}(\mathbf{x})$

20 : $\mathcal{V}$ samples $\mathbf{c} \leftarrow\!\!\$\ \mathbb{F}^{\log \frac{N'}{2k}}$ and sends it to $\mathcal{P}$

21 : $\mathcal{P}$ computes $S_{\mathbf{z}}(\mathbf{c}, \mathbf{x})$, $S_{\mathbf{w}}(\mathbf{c}, \mathbf{x})$

22 : $\mathcal{P}$ and $\mathcal{V}$ run sum-check protocol for $\sum_{\mathbf{i} \in \{0,1\}^{\log k}} S_{\mathbf{z}}(\mathbf{c}, \mathbf{i}) = 0$ and $\sum_{\mathbf{i} \in \{0,1\}^{\log k}} T(\mathbf{i}) = \sigma$

23 : $\mathcal{P}$ and $\mathcal{V}$ run sum-check protocol for $\sum_{\mathbf{i} \in \{0,1\}^{\log k}} S_{\mathbf{w}}(\mathbf{c}, \mathbf{i}) = 0$

24 : $\mathcal{P}$ sends $\widetilde{G}(\mathbf{c}, \mathbf{b})$, $\widetilde{E}(\mathbf{c}, \mathbf{b})$, $\widetilde{y}_{\mathbf{z}}(\mathbf{b})$, $\widetilde{y}_{\mathbf{w}}(\mathbf{b})$ to $\mathcal{V}$

25 : $\mathcal{V}$ computes $\widetilde{l}_{\mathbf{z}}(\mathbf{b})$, $\widetilde{r}_{\mathbf{z}}(\mathbf{b})$, and $\widetilde{r}_{\mathbf{w}}(\mathbf{b})$

26 : $\mathcal{V}$ checks $S_{\mathbf{z},\log k}(b_{\log k}) \stackrel{?}{=} \widetilde{G}(\mathbf{c}, \mathbf{b})\widetilde{y}_{\mathbf{z}}(\mathbf{b}) - \widetilde{E}(\mathbf{c}, \mathbf{b})\widetilde{r}_{\mathbf{z}}(\mathbf{b})$ and $T_{\log k}(b_{\log k}) \stackrel{?}{=} \widetilde{l}_{\mathbf{z}}(\mathbf{b})\widetilde{y}_{\mathbf{z}}(\mathbf{b})$

27 : $\mathcal{V}$ checks $S_{\mathbf{w},\log k}(b_{\log k}) \stackrel{?}{=} \widetilde{G}(\mathbf{c}, \mathbf{b})\widetilde{y}_{\mathbf{w}}(\mathbf{b}) - \widetilde{E}(\mathbf{c}, \mathbf{b})\widetilde{r}_{\mathbf{w}}(\mathbf{b})$

28 : $\mathcal{V}$ samples $a \leftarrow\!\!\$\ \mathbb{F}$ and sends it to $\mathcal{P}$

29 : $\mathcal{P}$ and $\mathcal{V}$ set $\mathbf{z}' = (\mathbf{c}||\mathbf{b}||a) \in \mathbb{F}^{\log N'}$

30 : $\mathcal{P}$ and $\mathcal{V}$ compute

$$\mathsf{cm}_{\mathbf{C}'} = \mathsf{Merkle.CM}(\mathsf{cm}_{\mathbf{G}[l_1:]}, \ldots, \mathsf{cm}_{\mathbf{G}[l_\ell:]}, \underbrace{0, \cdots, 0}_{\frac{N'}{2k}-\ell-1}, \mathsf{cm}_{\mathbf{y}_{\mathbf{z}}}, \mathsf{cm}_{\mathbf{E}[l_1:]}, \ldots, \mathsf{cm}_{\mathbf{E}[l_\ell:]}, \underbrace{0, \cdots, 0}_{\frac{N'}{2k}-\ell-1}, \mathsf{cm}_{\mathbf{y}_{\mathbf{w}}})$$

31 : $\mathcal{P}$ and $\mathcal{V}$ compute $\sigma' = (1-a)\left(\widetilde{G}(\mathbf{c}, \mathbf{b}) + \prod_{i=1}^{\log \frac{N'}{2k}} c_i \cdot \widetilde{y}_{\mathbf{z}}(\mathbf{b})\right) + a\left(\widetilde{E}(\mathbf{c}, \mathbf{b}) + \prod_{i=1}^{\log \frac{N'}{2k}} c_i \cdot \widetilde{y}_{\mathbf{w}}(\mathbf{b})\right)$

32 : $\mathcal{P}$ and $\mathcal{V}$ run LiLAC-Reduce$(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}'}, \mathsf{cm}_{\mathbf{D}'}, \mathbf{z}', \mathbf{z}', \sigma'; \mathbf{C}', \mathbf{E}')$

Fig. 4: LiLAC-Reduce

Let $\mathsf{rn} = \min\{i \mid N_i \leq N_{i+1}\}$ where $N_1 = N$, $N_{i+1} = \psi(2(\ell+1)\sqrt{N_i})$, and ℓ is a testing parameter. Our reduction process is executed rn rounds, and in the i-th reduction, the size of the polynomial decreases from N_i to $\psi(2(\ell+1)\sqrt{N_i})$. Each reduction involves a verification cost and proof generation of $\mathcal{O}(\log N_i)$ for the tensor IOPP and the Membership Proof Phase. Therefore, the total verification time and proof size across all reductions are $\sum \mathcal{O}(\log N_i) = \sum \mathcal{O}(\log N^{\frac{1}{2^i}}) = \mathcal{O}(\log N)$.

Let us examine the code provided in Figure 4 line by line to analyze its functionality. For simplicity, we assume the code rate ρ is fixed at $\frac{1}{2}$.[6]

MLPCS evaluation. (lines 1-3) If the number of coefficients in the polynomial cannot be further reduced, the reduction process terminates, and the existing MLPCS is invoked to directly verify whether $f_N^{\mathbf{C}}(\mathbf{z}) = \sigma$. This verification utilizes a multilinear polynomial commitment scheme, such as Brakedown [GLS+23] (see Appendix A). For a sufficiently reduced N, this operation imposes only constant overhead. If $f_N^{\mathbf{C}}(\mathbf{z}) = \sigma$, the verifier outputs 1; otherwise, it outputs 0.

Folding. (lines 4-8) If the reduction is possible, the verifier and prover engage in two crucial assessments: the consistency test, utilizing the vector $\mathbf{z}$, and the proximity test, employing the vector $\mathbf{w}$. As shown in Figure 4, the steps related to the proximity test with $\mathbf{w}$ are specifically highlighted in blue.

The prover first performs the Kronecker product $\otimes_{i=r/2+1}^{r}(1-z_i, z_i)$ on the right half of the vector $\mathbf{z} \in \mathbb{F}^r$, producing the vector $\mathbf{r_z} \in \mathbb{F}^k$. Similarly, the prover computes the Kronecker product $\otimes_{i=r/2+1}^{r}(1-w_i, w_i)$ on the right half of the vector $\mathbf{w} \in \mathbb{F}^r$, resulting in the vector $\mathbf{r_w} \in \mathbb{F}^k$. Using the vectors $\mathbf{r_z}$ and $\mathbf{r_w}$, the prover proceeds to fold the coefficient matrix $\mathbf{C}$, subsequently calculating the vectors $\mathbf{y_z} = \mathbf{C} \cdot \mathbf{r_z}$ and $\mathbf{y_w} = \mathbf{C} \cdot \mathbf{r_w}$. The prover then constructs the Merkle trees corresponding to the vectors $\mathbf{y_z}$ and $\mathbf{y_w}$, denoted as $\mathsf{cm}_{\mathbf{y_z}}$ and $\mathsf{cm}_{\mathbf{y_w}}$, respectively, and sends these to the verifier.

Finally, the verifier recomputes the Merkle trees $\mathsf{cm}_{\mathbf{C}}$ and $\mathsf{cm}_{\mathbf{D}}$ to generate the Merkle tree of the codeword $\mathbf{E}$, denoted as $\mathsf{cm}_{\mathbf{E}}$.

Membership Proof. (lines 9-12) Given the random ℓ indices in the set $I = \{l_1, \cdots, l_\ell\}$, for each $l_t \in I$, the prover sends the Merkle commitment $\mathsf{cm}_{\mathbf{G}[l_t:]}$ for the l_t-th row $\mathbf{G}[l_t, :]$ of $\mathbf{G}$ and $\mathsf{cm}_{\mathbf{E}[l_t:]}$ for the l_t-th row $\mathbf{E}[l_t, :]$ of $\mathbf{E}$, along with the co-paths for each commitment, to the verifier. The verifier then uses the received commitments and co-paths to verify membership, ensuring that for all $t \in [\ell]$, $\mathsf{cm}_{\mathbf{G}[l_t:]} \in \mathsf{cm}_{\mathbf{G}}$ and $\mathsf{cm}_{\mathbf{E}[l_t:]} \in \mathsf{cm}_{\mathbf{E}}$.

Commit. (lines 13-15) During this phase, the prover constructs a new coefficient matrix $\mathbf{C}'$ with $N' = \psi(2(\ell+1)\sqrt{N})$ coefficients using the vectors $\mathbf{G}[l_1, :], \cdots, \mathbf{G}[l_\ell, :]$, $\mathbf{E}[l_1, :], \cdots, \mathbf{E}[l_\ell, :]$, along with the vectors $\mathbf{y_z}$ and $\mathbf{y_w}$. To

[6] When $\rho = \frac{1}{2}$, the verifier does not require additional verification steps to confirm that the codeword $\mathbf{E}$ is derived from the coefficient matrix $\mathbf{C}$, simplifying the scheme. For general code rates $\rho > 0$, where this verification is necessary, we provide a detailed explanation in Section 3.3.

fully utilize the tree structure of the Merkle commitment, the prover replicates the vectors $\mathbf{y_z}$ and $\mathbf{y_w}$ ℓ times, resulting in $\mathbf{C'}$:

$$\mathbf{C'} = \Big(\mathbf{G}[l_1,:],\ldots,\mathbf{G}[l_\ell:], \underbrace{0,\cdots,0}_{\frac{N'}{2}-k\ell-k}, \mathbf{y_z},$$
$$\mathbf{E}[l_1,:],\ldots,\mathbf{E}[l_\ell:], \underbrace{0,\cdots,0}_{\frac{N'}{2}-k\ell-k}, \mathbf{y_w}\Big) \in \mathbb{F}^{N'}$$

Subsequently, the prover encodes $\mathbf{C'}$ using the encoding function for next round $\mathsf{rn'}$, denoted as $\mathsf{Enc}_{\mathsf{rn'}}$, to compute the new codeword $\mathbf{E'} = \mathsf{Enc}_{\mathsf{rn'}}(\mathbf{C'})$.

Assuming that all encodings use systematic codes such as the Spielman code [Spi95] or the generalized Spielman code introduced in [GLS+23][7], the codeword $\mathbf{E'}$ resulting from encoding the matrix $\mathbf{C'}$ is represented as $\left[\frac{\mathbf{C'}}{\mathbf{D'}}\right] \in \mathbb{F}^{n'\times k'}$, where $\mathbf{D'}$ serves as the syndrome in $\mathbb{F}^{k'\times k'}$. Finally, the prover commits to $\mathbf{D'}$ using a Merkle commitment scheme, producing a commitment $\mathsf{cm}_{\mathbf{D'}}$, which is then sent to the verifier.

Multilinear Extension. (lines 16-19) In this phase, we verify the ℓ consistency checks, proximity checks, and the processes of encoding and folding by utilizing the sum-check protocol. Each matrix and vector involved is converted into a multilinear polynomial and structured within a sum-check framework.

The prover and verifier first perform the multilinear extension of the vectors $\mathbf{l_z} = \otimes_{i=1}^{r/2}(1-z_i, z_i)$, $\mathbf{r_z}$, and $\mathbf{r_w} \in \mathbb{F}^k$, thereby obtaining the multilinear polynomials $\widetilde{l}_{\mathbf{z}}(\mathbf{x})$, $\widetilde{r}_{\mathbf{z}}(\mathbf{x})$, and $\widetilde{r}_{\mathbf{w}}(\mathbf{x}) \in \mathbb{F}[x_1,\ldots,x_{\log k}]$. Additionally, the prover performs the multilinear extension of the vectors $\mathbf{y_z}$, $\mathbf{y_w} \in \mathbb{F}^k$ and the matrices $\mathbf{G}[I,:]$ and $\mathbf{E}[I,:] \in \mathbb{F}^{\ell\times k}$, resulting in the multilinear polynomials $\widetilde{y}_{\mathbf{z}}(\mathbf{x})$, $\widetilde{y}_{\mathbf{w}}(\mathbf{x}) \in \mathbb{F}[x_1,\ldots,x_{\log k}]$, $\widetilde{G}(\mathbf{y},\mathbf{x})$, and $\widetilde{E}(\mathbf{y},\mathbf{x}) \in \mathbb{F}[y_1,\ldots,y_{\log\frac{N'}{2k}},x_1,\ldots,x_{\log k}]$.

Due to the commutative property of encoding and folding (see Proposition 1), these multilinear polynomials satisfy the following conditions for all $\mathbf{j} \in \{0,1\}^{\log\frac{N'}{2k}}$:

$$\sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{G}(\mathbf{j},\mathbf{i})\widetilde{y}_{\mathbf{z}}(\mathbf{i}) = \sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{E}(\mathbf{j},\mathbf{i})\widetilde{r}_{\mathbf{z}}(\mathbf{i}) \quad \text{(Consistency)}$$

$$\sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{G}(\mathbf{j},\mathbf{i})\widetilde{y}_{\mathbf{w}}(\mathbf{i}) = \sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{E}(\mathbf{j},\mathbf{i})\widetilde{r}_{\mathbf{w}}(\mathbf{i}) \quad \text{(Proximity)}$$

Furthermore, since the folding of the vector $\mathbf{y_z}$ with $\mathbf{l_z}$ equals σ, we also have:

$$\sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{l}_{\mathbf{z}}(\mathbf{i})\widetilde{y}_{\mathbf{z}}(\mathbf{i}) = \sigma \quad \text{(Tensor product)}$$

[7] Reed-Solomon (RS) codes [RS60] could also be used; however, this increases the prover's cost to $\mathcal{O}(N\log N)$ due to the Fast Fourier Transform and removes the scheme's field-agnostic property.

Using these identities, the prover defines the following multivariable polynomials:

$$S_{\mathbf{z}}(\mathbf{y},\mathbf{x}) := \widetilde{G}(\mathbf{y},\mathbf{x})\widetilde{y}_{\mathbf{z}}(\mathbf{x}) - \widetilde{E}(\mathbf{y},\mathbf{x})\widetilde{r}_{\mathbf{z}}(\mathbf{x}) \quad \text{(Consistency)}$$
$$S_{\mathbf{w}}(\mathbf{y},\mathbf{x}) := \widetilde{G}(\mathbf{y},\mathbf{x})\widetilde{y}_{\mathbf{w}}(\mathbf{x}) - \widetilde{E}(\mathbf{y},\mathbf{x})\widetilde{r}_{\mathbf{w}}(\mathbf{x}) \quad \text{(Proximity)}$$
$$T_{\mathbf{z}}(\mathbf{x}) := \widetilde{l}_{\mathbf{z}}(\mathbf{x})\widetilde{y}_{\mathbf{z}}(\mathbf{x}) \quad \text{(for tensor product)}$$

Sum-check. (lines 20-27) The verifier samples and sends the vector $\mathbf{c} \in \mathbb{F}^{\log \frac{N'}{2k}}$ to the prover. In response, the prover calculates the polynomials $S_{\mathbf{z}}(\mathbf{c},\mathbf{x})$ and $S_{\mathbf{w}}(\mathbf{c},\mathbf{x})$ using the provided vector $\mathbf{c}$. For the consistency check and proximity test, the verifier employs the sum-check protocol to verify the following conditions:

$$\sum_{\mathbf{i}\in\{0,1\}^{\log k}} S_{\mathbf{z}}(\mathbf{c},\mathbf{i}) = 0, \qquad \sum_{\mathbf{i}\in\{0,1\}^{\log k}} S_{\mathbf{w}}(\mathbf{c},\mathbf{i}) = 0,$$
$$\sum_{\mathbf{i}\in\{0,1\}^{\log k}} T_{\mathbf{z}}(\mathbf{i}) = \sigma.$$

Unlike the original sum-check protocol, our scheme does not require the verifier to directly compute the final polynomial evaluations, such as $S_{\mathbf{z},\log k}(b_{\log k}) = S_{\mathbf{z}}(\mathbf{c},\mathbf{b})$, $S_{\mathbf{w},\log k}(b_{\log k}) = S_{\mathbf{w}}(\mathbf{c},\mathbf{b})$, and $T_{\mathbf{z},\log k}(b_{\log k}) = T_{\mathbf{z}}(\mathbf{b})$. Instead, the verifier receives these computed values—$\widetilde{G}(\mathbf{c},\mathbf{b})$, $\widetilde{E}(\mathbf{c},\mathbf{b})$, $\widetilde{y}_{\mathbf{z}}(\mathbf{b})$, and $\widetilde{y}_{\mathbf{w}}(\mathbf{b})$—directly from the prover for verification. Here, the vector $\mathbf{b}$ is composed of all the challenges $b_1, \cdots, b_{\log k}$, which are randomly chosen by the verifier in the sum-check protocol and sent to the prover in each round.

Additionally, the verifier calculates $\widetilde{l}_{\mathbf{z}}(\mathbf{b})$, $\widetilde{r}_{\mathbf{z}}(\mathbf{b})$, and $\widetilde{r}_{\mathbf{w}}(\mathbf{b})$ independently to confirm the following conditions:

$$S_{\mathbf{z},\log k}(b_{\log k}) = \widetilde{G}(\mathbf{c},\mathbf{b})\widetilde{y}_{\mathbf{z}}(\mathbf{b}) - \widetilde{E}(\mathbf{c},\mathbf{b})\widetilde{r}_{\mathbf{z}}(\mathbf{b}),$$

$$S_{\mathbf{w},\log k}(b_{\log k}) = \widetilde{G}(\mathbf{c},\mathbf{b})\widetilde{y}_{\mathbf{w}}(\mathbf{b}) - \widetilde{E}(\mathbf{c},\mathbf{b})\widetilde{r}_{\mathbf{w}}(\mathbf{b}),$$

$$T_{\mathbf{z},\log k}(b_{\log k}) = \widetilde{l}_{\mathbf{z}}(\mathbf{b})\widetilde{y}_{\mathbf{z}}(\mathbf{b}).$$

This methodology enhances the computational efficiency of the verifier, but it imposes a crucial requirement: the verifier must rigorously ensure the correctness of each of the $\widetilde{G}$, $\widetilde{E}$, $\widetilde{y}_{\mathbf{z}}$, and $\widetilde{y}_{\mathbf{w}}$.

Invoke for the Next Round. (lines 28-32) To address the aforementioned issue, we define a new function $f_{N'}^{\mathbf{C}'}$ where $N' = \psi(2(\ell+1)k)$.

The verifier sends a scalar $a \in \mathbb{F}$ to the prover. Both the prover and verifier then concatenate the vectors $a \in \mathbb{F}$, $\mathbf{b} \in \mathbb{F}^{\log k}$, and $\mathbf{c} \in \mathbb{F}^{\log \frac{N'}{2k}}$ to define a new vector $\mathbf{z}' = (\mathbf{c}||\mathbf{b}||a) \in \mathbb{F}^{N'}$ and set $\mathbf{G}' = \mathbf{G}_{\mathsf{rn}'}$. Next, the prover and verifier use the commitments $\mathsf{cm}_{\mathbf{G}[l_1:]}$, …, $\mathsf{cm}_{\mathbf{G}[l_\ell:]}$, $\mathsf{cm}_{\mathbf{y}_{\mathbf{z}}}$, $\mathsf{cm}_{\mathbf{E}[l_1:]}$, …, $\mathsf{cm}_{\mathbf{E}[l_\ell:]}$, and $\mathsf{cm}_{\mathbf{y}_{\mathbf{w}}}$ to compute the commitment $\mathsf{cm}_{\mathbf{C}'}$ for the subsequent round.

Similar to the approach described in line 13, to match the size of the Merkle tree (see Figure 2), we pad with $\frac{N'}{2k} - \ell - 1$ zeros, thus forming:

$$\mathsf{cm}_{\mathbf{C}'} = \mathsf{Merkle.CM}(\mathsf{cm}_{\mathbf{G}[l_1:]}, \ldots, \mathsf{cm}_{\mathbf{G}[l_\ell:]}, \underbrace{0, \cdots, 0}_{\frac{N'}{2k}-\ell-1}, \mathsf{cm}_{\mathbf{y_z}},$$
$$\mathsf{cm}_{\mathbf{E}[l_1:]}, \ldots, \mathsf{cm}_{\mathbf{E}[l_\ell:]}, \underbrace{0, \cdots, 0}_{\frac{N'}{2k}-\ell-1}, \mathsf{cm}_{\mathbf{y_w}}).$$

Finally, using the scalar a, the prover and verifier calculate the scalar σ' as follows:

$$\sigma' = (1-a)\left(\widetilde{G}(\mathbf{c},\mathbf{b}) + \prod_{i=1}^{\log \frac{N'}{2k}} c_i \cdot \widetilde{y}_{\mathbf{z}}(\mathbf{b})\right) + a\left(\widetilde{E}(\mathbf{c},\mathbf{b}) + \prod_{i=1}^{\log \frac{N'}{2k}} c_i \cdot \widetilde{y}_{\mathbf{w}}(\mathbf{b})\right)$$

(thus, σ' is equal to $f_{N'}^{\mathbf{C}'}(\mathbf{z}')$). The prover and verifier then proceed by invoking the LiLAC-Reduce with the new input $(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}'}, \mathsf{cm}_{\mathbf{D}'}, \mathbf{z}', \mathbf{w}' = \mathbf{z}', \sigma'; \mathbf{C}', \mathbf{E}')$ with the next round rn'.

Theorem 1. *For an $[n, k, d]$-linear code $\mathcal{C}$, with the testing parameter ℓ and relative distance δ, the LiLAC-Reduce (for the first round) is both complete and sound, with soundness error given by $\epsilon_1 = \max\{\epsilon_{Reduce}, \epsilon_{MLPCS}\}$, where*

$$\epsilon_{Reduce} = \mathcal{O}\left(\frac{\ell \cdot \log n + d^2 + 2\log k}{|\mathbb{F}|} + \left(1 - \frac{\delta}{2}\right)^{\ell}\right)$$

and ϵ_{PCS} denotes the soundness error of the multilinear polynomial commitment scheme utilized in the MLPCS evaluation phase.

Proof. Refer to the Appendix B for the proof.

Recursive rounds. From the second reduction onward, the process is almost identical to the first reduction, with the key difference merging the proximity and consistency test. This is because the multilinear polynomial, reduced in size after the first reduction, already includes both proximity and consistency tests, allowing verification of the initial polynomial by performing tensor IOPP (consistency and tensor product) on this polynomial.

Specifically, while two different vectors $\mathbf{z}$ and $\mathbf{w}$ were used for consistency and proximity, respectively, from the second reduction onward, both the consistency and proximity tests are conducted using the same vector, $\mathbf{z}$ and $\mathbf{z}$.

Theorem 2. *Let $2 \leq i \leq \mathsf{rn}$ where rn denotes the number of rounds during which the number of coefficients is reduced. For an $[n_i, k_i, d_i]$-linear code $\mathcal{C}_i$ with testing parameter ℓ_i and relative distance δ_i, the LiLAC-Reduce (for subsequent rounds) is both complete and sound, with soundness error ϵ_2 at the i-th recursive round given by $\max\{\epsilon_{Reduce}, \epsilon_{MLPCS}\}$, where*

$$\epsilon_{Reduce} = \mathcal{O}\left(\frac{\ell_i \cdot \log n_i + d_i^2 + 2\log k_i}{|\mathbb{F}|} + \left(1 - \frac{\delta_i}{2}\right)^{\ell_i}\right),$$

and ϵ_{MLPCS} denotes the soundness error of the multilinear polynomial commitment employed in the MLPCS evaluation phase.

Proof. This follows directly from Theorem 1.

Theorem 3. *Let N denote the size of the coefficients, such that $N = k^2$. For an $[n, k, d]$-linear code $\mathcal{C}$, with testing parameter ℓ and relative distance δ, there exists a field-agnostic multilinear polynomial commitment scheme characterized by the following parameters:*

- *The prover performs $\mathcal{O}(N)$ field operations;*
- *The proof length consists of $\mathcal{O}(\log N)$ field elements;*
- *The verifier requires $\mathcal{O}(\log N)$ field operations;*
- *The soundness error is bounded by*

$$\mathcal{O}\left(\frac{\ell \cdot \log n + d^2 + \log N}{|\mathbb{F}|} + \left(1 - \frac{\delta}{2}\right)^{\ell}\right) + \epsilon_{MLPCS}.$$

Proof. The completeness of the LiLAC protocol follows directly from Theorems 1 and 2. Our LiLAC procedure performs LiLAC-Reduce (first round), which subsequently invokes LiLAC-Reduce (subsequent rounds) $(\mathsf{rn} - 1)$ times, where rn is the that is at most $\log \log N$ and regarded as constant. Therefore, the total soundness error for all reductions is

$$\epsilon_1 + \sum_{i=2}^{\mathsf{rn}} \epsilon_2 < \mathsf{rn} \cdot \epsilon_1 = \mathcal{O}\left(\frac{\ell \cdot \log n + d^2 + \log N}{|\mathbb{F}|} + \left(1 - \frac{\delta}{2}\right)^{\ell}\right).$$

Upon completing all rounds of reduction and reaching the maximum rn, the MLPCS protocol is applied, and thus the total soundness error of LiLAC is bounded by

$$\mathcal{O}\left(\frac{\ell \cdot \log n + d^2 + \log N}{|\mathbb{F}|} + \left(1 - \frac{\delta}{2}\right)^{\ell}\right) + \epsilon_{\mathrm{MLPCS}}.$$

3.3 Generalization and Optimization of LiLAC

Extension to Non-systematic Codes. We extend the LiLAC-Reduce process to handle cases involving non-systematic codes. In the LiLAC-Reduce process, the verifier must ensure that the codeword $\mathbf{E}$ is correctly computed from the coefficient matrix $\mathbf{C}$. Specifically, the verifier must confirm the relation $\mathbf{E} = \mathbf{G} \cdot \mathbf{C}$, where $\mathbf{G}$ is the encoding matrix. While systematic codes allow this verification using the commitments $\mathsf{cm}_{\mathbf{C}}$ and $\mathsf{cm}_{\mathbf{D}}$, non-systematic codes do not separate $\mathbf{E}$ into two distinct parts ($\mathbf{C}$ and $\mathbf{D}$), making direct verification infeasible. Instead, the verifier indirectly ensures that $\mathbf{E} = \mathbf{G} \cdot \mathbf{C}$ by verifying that the matrix $\mathbf{C}$ has been folded correctly. This is achieved by verifying the relation $\mathbf{E} \cdot \mathbf{r}_{\mathbf{z}} = \mathbf{G} \cdot \mathbf{y}_{\mathbf{z}}$

via a sum-check protocol and additionally verifying $\mathbf{y_z} = \mathbf{C} \cdot \mathbf{r_z}$. Together, these checks guarantee that:

$$\mathbf{E} \cdot \mathbf{r_z} = \mathbf{G} \cdot (\mathbf{C} \cdot \mathbf{r_z}).$$

Thus, during LiLAC-Reduce, the algorithm incorporates the additional check $\mathbf{y_z} = \mathbf{C} \cdot \mathbf{r_z}$. As part of the proximity check, a similar verification for $\mathbf{w}$, $\mathbf{y_w} = \mathbf{C} \cdot \mathbf{r_w}$, is required. Since the checks for $\mathbf{z}$ and $\mathbf{w}$ are symmetric, it suffices to describe the process for $\mathbf{z}$.

This additional verification is efficiently handled using the sum-check protocol, similar to the consistency check in LiLAC-Reduce. Specifically, we define the multilinear polynomial $S'_{\mathbf{z}}(\mathbf{y}, \mathbf{x}) \in \mathbb{F}[y_1, \cdots, y_{\log k}, x_1, \cdots, x_{\log k}]$ as:

$$\widetilde{y}_{\mathbf{z}}(\mathbf{y}) - \sum_{\mathbf{x} \in \{0,1\}^{\log k}} \widetilde{C}(\mathbf{y}, \mathbf{x}) \cdot \widetilde{r}_{\mathbf{z}}(\mathbf{x}),$$

where $\widetilde{C}$ is the multilinear extension of $\mathbf{C}$. Then, the prover and verifier run an additional sum-check protocol for:

$$\sum_{\mathbf{i} \in \{0,1\}^{\log k}} S'_{\mathbf{z}}(\mathbf{c}', \mathbf{i}) = 0,$$

where $\mathbf{c}' \leftarrow\!\!\$\ \mathbb{F}^{\log k}$ is a random challenge. This ensures, with high probability, that $\mathbf{E} = \mathbf{G} \cdot \mathbf{C}$, thereby extending the verification process to non-systematic codes.

Generalization for Arbitrary Code Rate. When the generic code rate ρ is applied (i.e., $n_i \neq 2k_i$), the verifier cannot derive $\mathsf{cm}_{\mathbf{E}}$ solely from $\mathsf{cm}_{\mathbf{C}}$ and $\mathsf{cm}_{\mathbf{D}}$. This is because, for $n_i \neq 2k_i$, the sizes of $\mathbf{C} \in \mathbb{F}^{k \times k}$ and $\mathbf{D} \in \mathbb{F}^{(n-k) \times k}$ differ, resulting in $\mathsf{Merkle.CM}(\mathsf{Merkle.CM}(\mathbf{C}), \mathsf{Merkle.CM}(\mathbf{D})) \neq \mathsf{Merkle.CM}(\mathbf{E})$. In this case, instead of transmitting $\mathsf{cm}_{\mathbf{D}}$ as described in Figure 4, the prover sends the commitment $\mathsf{cm}_{\mathbf{E}}$ along with the co-path information of $\mathsf{cm}_{\mathbf{C}}$. The verifier then uses this co-path to confirm that $\mathsf{cm}_{\mathbf{C}} \in \mathsf{cm}_{\mathbf{E}}$, ensuring that $\mathbf{C}$ is part of $\mathbf{E}$ and that the codeword $\mathbf{E}$ was generated from $\mathbf{C}$.

In Eval, the prover does not compute matrix $\mathbf{D}$ or its commitment $\mathsf{cm}_{\mathbf{D}}$. Instead, the prover transmits the commitment $\mathsf{cm}_{\mathbf{E}}$ of codeword $\mathbf{E}$ along with the co-path information of $\mathsf{cm}_{\mathbf{C}}$ to the verifier. The verifier uses this co-path to confirm that $\mathsf{cm}_{\mathbf{C}} \in \mathsf{cm}_{\mathbf{E}}$. Subsequently, the prover and verifier set the input $(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}}, \mathsf{cm}_{\mathbf{E}}, \mathbf{z}, \mathbf{w}, \sigma; \mathbf{C}, \mathbf{E})$ and proceed with the LiLAC-Reduce protocol, using cm_E in place of $\mathsf{cm}_{\mathbf{D}}$.

In LiLAC-Reduce, since the verifier already holds $\mathsf{cm}_{\mathbf{E}}$, it does not need to compute $\mathsf{cm}_{\mathbf{E}}$ again. Similarly to Eval, the prover computes $\mathsf{cm}_{\mathbf{E}'}$ instead of $\mathsf{cm}_{\mathbf{D}'}$ and sends it to the verifier. With this, the prover and verifier set the new input $(\mathsf{pp}, \mathsf{cm}_{\mathbf{C}'}, \mathsf{cm}_{\mathbf{E}'}, \mathbf{z}', \mathbf{w}', \sigma'; \mathbf{C}', \mathbf{E}')$ and proceed recursively with the protocol LiLAC-Reduce.

Code Selection for Optimizing ℓ. Optimizing the parameter N' requires minimizing the testing parameter ℓ, which depends on the relative distance δ of

the error-correcting code. Our approach balances the trade-offs between proving time, proof size, and verification time by strategically selecting codes based on the current size of N. Initially, we use generalized Spielman codes [GLS+23, XZS22], which operate in linear time, to minimize proving time while efficiently reducing N to a manageable size. Once N is sufficiently small, we transition to Reed-Solomon codes [BSBHR18, ZCF23, ACFY24a], which enable a smaller ℓ, further reducing N'.

This two-stage approach leverages the strengths of both codes. Spielman codes prioritize prover efficiency during the early stages when N is large, ensuring that the overall proving time remains manageable. Reed-Solomon codes, applied once N has decreased, trade a slight increase in proving time (due to logarithmic complexity) for significant reductions in proof size and verification time.

By adapting the code choice to the current size of N, this method achieves a more balanced optimization across all three metrics: proving time, proof size, and verification time.

Dynamic Adjustment of Code Rates. Proving time, verifying time, and proof size in code-based PCS are closely interdependent, as these schemes rely on error-correcting codes (ECC) to construct proofs. The proximity test in PCS depends on the error-correcting capability (or minimum Hamming distance) of the ECC, which determines the number of points sampled. A code with stronger error-correcting capability reduces the number of points sampled, thereby decreasing verifying time and proof size, but at the cost of increased proving time.

To balance these trade-offs, we dynamically adjust the code rate ρ based on the size of N. When N is large, using codes with higher rates (e.g., $\rho = 1/2$) minimizes proving time by reducing the cost per coefficient. As N decreases, codes with lower rates (e.g., $\rho = 1/64$ or $\rho = 1/256$) are applied. While these increase proving time with a complexity of $\mathcal{O}(N)$, the smaller N mitigates this impact. Additionally, the improved error-correcting capability of these codes allows ℓ to be minimized, leading to more significant reductions in proof size and verification time. In our proposed MLPCS, each round operates independently, allowing for dynamic adjustment of ρ based on N.

If a code with ρ code rate is applied to N coefficients, the encoding time increases to $\frac{1}{\rho} \cdot N$, while the number of sampling points for the proximity test decreases by a factor of $-\log \rho$. Consequently, the verifying time and proof size are reduced to $-\log \rho \cdot \log N$, as each sampling point requires $\log N$ membership proofs. Proving time follows a complexity of $\mathcal{O}\left(\frac{1}{\rho} \cdot N\right)$, while verifying time and proof size are $\mathcal{O}\left(\frac{\log N}{-\log \rho}\right)$. By dynamically tuning ρ, we achieve a balanced reduction across all metrics, leveraging the trade-off between encoding time and the reductions in verifying time and proof size.

For example, applying a code rate of $1/4$ for 2^{30} and $1/2$ for 2^{20} results in a proving time of $4 \cdot 2^{30} + 2 \cdot 2^{20} > 2^{32}$ and a verifying time and proof size of $\frac{30}{2} + 20 = 35$. Alternatively, applying code rates of $1/2$ and $1/32$ results in a proving time of $2 \cdot 2^{30} + 32 \cdot 2^{20} < 2^{32}$ and a verifying time and proof size of $30 + \frac{20}{5} = 34$, showing a reduction in both metrics compared to the initial

configuration. Therefore, when N is small, using low code rates significantly reduces verification time and proof size, with only a small increase in proving time.

3.4 Complexity of LiLAC

In this section, we analyze the asymptotic complexity of LiLAC. The protocol consists of two main phases: Setup and Eval. The Eval phase recursively invokes LiLAC-Reduce for the first round, followed by LiLAC-Reduce for the subsequent rounds across $\mathsf{rn}-1$ iterations. We will discuss the asymptotic costs for both the prover and the verifier in each sub-protocol, as well as the size of the resulting proof.

For convenience, in this section, we will refer to the two types of reduction algorithms, LiLAC-Reduce for the first round and LiLAC-Reduce for the subsequent rounds, as LiLAC-Reduce$_1$ and LiLAC-Reduce$_2$, respectively. The symbols $\mathbb{F}$ and H denote field multiplication and hash operation, respectively.

Preprocessing. In the Setup, the party computes the Merkle tree commitments $\mathsf{cm}_{\mathbf{G}_i}$ for each rn. Given that each matrix $\mathbf{G}_i$ has dimensions $n_i \times k_i$, the total preprocessing cost is $\sum_{i=1}^{\mathsf{rn}} n_i k_i = \mathcal{O}(N)\mathsf{H}$.

Prover's cost. During the Eval, the prover encodes $\mathbf{C} \in \mathbb{F}^N$ using the encoding function Enc, with a cost of $nk\mathbb{F}$. To compute the Merkle tree commitment for the submatrix $\mathbf{D}$, the prover performs $(n-k) \times k$ hash operations, which amounts to $N\mathsf{H}$. Additionally, the evaluation value σ is computed with $\log N$ field operations.

In the LiLAC-Reduce$_1$, the prover calculates the vectors $\mathbf{r_z}$, $\mathbf{r_w}$, $\mathbf{y_z}$, and $\mathbf{y_w}$ at a cost of $2\sqrt{N}\mathbb{F}$ and $2N\mathbb{F}$. The commitments $\mathsf{cm}_{\mathbf{y_z}}$, $\mathsf{cm}_{\mathbf{y_w}}$, $\mathsf{cm}_{\mathbf{E}[l_t:]}$, and $\mathsf{cm}_{\mathbf{G}[l_t:]}$ require $\sqrt{N}$ and $\ell\sqrt{N}$ hash operations. To generate the codeword $\mathbf{E}'$ and its submatrix commitment $\mathsf{cm}_{\mathbf{D}'}$ for the next round, the prover incurs costs of $N_i\mathbb{F}$ and $N_i\mathsf{H}$. The sum-check protocol adds a cost of $N\mathbb{F}$, while the computations of $\widetilde{G}(\mathbf{c}, \mathbf{b})$, $\widetilde{E}(\mathbf{c}, \mathbf{b})$, $\widetilde{y}_{\mathbf{z}}(\mathbf{b})$, and $\widetilde{y}_{\mathbf{w}}(\mathbf{b})$ require $N'\mathbb{F}$ and $2\sqrt{N}\mathbb{F}$.

Additionally, computing the commitment for the new input $\mathbf{C}'$ and its evaluation σ' in the next round requires $(2\ell+2)\mathsf{H}$ and $\left(\log \frac{2N'}{k} + 4\right)\mathbb{F}$, respectively. Therefore, the total cost in the first round is $\mathcal{O}(N)\mathbb{F}$ and $\mathcal{O}(N)\mathsf{H}$. In the LiLAC-Reduce$_2$, the operations are similar to those in LiLAC-Reduce$_1$, excluding the proximity test for the vector $\mathbf{w}$. Thus, in each round i, the total cost is $\mathcal{O}(N_i)\mathbb{F}$ and $\mathcal{O}(N_i)\mathsf{H}$.

Overall, the total prover cost in LiLAC is $\mathcal{O}(N)\mathbb{F} + \sum_{i=1}^{\mathsf{rn}} \mathcal{O}(N_i)\mathbb{F} = \mathcal{O}(N)\mathbb{F}$, and $\mathcal{O}(N)\mathsf{H} + \sum_{i=1}^{\mathsf{rn}} \mathcal{O}(N_i)\mathsf{H} = \mathcal{O}(N)\mathsf{H}$.

Communication cost. In the Eval, the prover sends the commitment $\mathsf{cm}_{\mathbf{D}} \in \mathbb{F}$ and the evaluation $\sigma \in \mathbb{F}$ to the verifier. During the reduction process, the prover transmits $(2+2\ell)$ Merkle tree commitments, specifically $\mathsf{cm}_{\mathbf{y_z}}$, $\mathsf{cm}_{\mathbf{y_w}}$, $\mathsf{cm}_{\mathbf{G}[l_t:]}$, and $\mathsf{cm}_{\mathbf{E}[l_t:]} \in \mathbb{F}$, in each round. Additionally, co-path data for $\mathsf{cm}_{\mathbf{G}[l_t:]}$ and $\mathsf{cm}_{\mathbf{E}[l_t:]}$ is provided, with each data point sized at $\log n_i$, totaling $2\ell \log n_i$ field

elements. The prover also sends the commitment $\mathsf{cm}_{\mathbf{D}'} \in \mathbb{F}$ for the next round's submatrix, along with $2\log\sqrt{N_i}$ field elements during the sum-check protocol. Finally, the prover transmits the values $\widetilde{G}(\mathbf{c},\mathbf{b})$, $\widetilde{E}(\mathbf{c},\mathbf{b})$, $\widetilde{y}_{\mathbf{z}}(\mathbf{b})$, and $\widetilde{y}_{\mathbf{w}}(\mathbf{b}) \in \mathbb{F}$ for the evaluation step of the sum-check protocol.

Thus, the total communication cost for the prover is $\sum_{i=1}^{\mathsf{rn}}(7+2\ell+2\ell\log n_i + 2\log\sqrt{N_i})\mathbb{F} = \mathcal{O}(\log N)\mathbb{F}$.

Verifer's cost. In the LiLAC-Reduce$_1$, the verifier uses the commitments $\mathsf{cm}_{\mathbf{C}}$ and $\mathsf{cm}_{\mathbf{D}}$ to compute the Merkle commitment $\mathsf{cm}_{\mathbf{E}}$ with a single hash operation. The verifier then performs a membership proof, which requires $\log n\mathsf{H}$ per component, resulting in a total of $2\ell\log n$ hash operations. During the sum-check protocol, the verifier performs $2\log\sqrt{N}$ field operations and verifies the evaluations $S_{\mathbf{z},\log k}$, $S_{\mathbf{w},\log k}$, and $T_{\log k}$ using $\widetilde{l}_{\mathbf{z}}(\mathbf{b})$, $\widetilde{r}_{\mathbf{z}}(\mathbf{b})$, and $\widetilde{r}_{\mathbf{w}}(\mathbf{b})$, each requiring $\log\sqrt{N}$ field operations. Finally, the verifier computes the commitments $\mathsf{cm}_{\mathbf{C}'}$ and σ' for the next round, requiring $(2\ell+2)\mathsf{H}$ and $\left(\log\frac{2N'}{k}+4\right)\mathbb{F}$, respectively. Thus, the verifier's cost in LiLAC-Reduce$_1$ is $\mathcal{O}(\log N)$ field operations. Similarly, in the LiLAC-Reduce$_2$, the verifier incurs a cost of $\mathcal{O}(\log N_i)$ field operations.

Therefore, the total cost for the verifier is $\sum_{i=1}^{\mathsf{rn}}\mathcal{O}(\log N_i)\mathbb{F} = \mathcal{O}(\log N)\mathbb{F}$.

4 Experiments

In this section, we present the implementation and performance evaluation of our proposed scheme, LiLAC, and compare it with existing polynomial commitment schemes, including Brakedown [GLS+23], Basefold, BasefoldFri [ZCF23], Orion [XZS22], FRI [BSBHR18], STIR [ACFY24a], and WHIR [ACFY24b]. Our implementation of LiLAC is written in C++ and builds upon the open-source implementation of Orion [Ori], incorporating generalized Spielman codes with an expander graph-based error correction code. The hash function used is SHA3. All experiments were conducted on an Intel Xeon Gold 6242R CPU @ 3.10GHz with 256GB memory running Ubuntu 22.

For performance evaluation, we tested polynomials of various sizes, specifically ranging from $N = 2^{22}$ to 2^{30}. For each case, we measured three core metrics: proving time, proof size, and verification time. In our experiments, proving time is defined as the total time required for commitment, encoding, and opening, with the exception of FRI, and STIR. Note that FRI, and STIR only perform the proximity test.

In this implementation, we propose two models. The first model, LiLAC (Brakedown / Basefold), employs the Spielman code combined with Brakedown or Basefold MLPCS to achieve linear proving time and field-agnostic properties. The second model, LiLAC(Shockwave), utilizes Shockwave MLPCS which is identical to Brakedown MLPCS with Reed-Solomon codes, achieving smaller proof sizes and faster proving times, albeit without field-agnosticism.

Our experiments are categorized based on field-agnostic requirements. For field-agnostic MLPCS, we compared and analyzed Brakedown, Basefold, and

our LiLAC(Brakedown/Basefold). For field-specific MLPCS, we compared BasefoldFri, Orion, and WHIR with our LiLAC(Shockwave). Additionally, while FRI and STIR are not MLPCS, we included their experimental results for broader comparison. To evaluate the efficiency of the reduction process in LiLAC (Brakedown/Basefold) and LiLAC(Shockwave), we measured proving time and proof size at each reduction round. Cases marked as "N/A" indicate instances where results could not be obtained due to insufficient memory on our 256GB system.

4.1 Field-agnostic MLPCS comparison

Parameters. We evaluate the performance of field-specific polynomial commitment schemes over 128-bit fields, setting the security parameter $\lambda = 128$ for all protocols. For Basefold, we used the parameters from the open-source implementation of Basefold [Bas]. The code rate is 1/2, and the query counts are 808, 971, and 1358 for 2^{22}, 2^{24}, and 2^{26}, respectively. For 2^{28}, since the parameter is not provided, we set the query count to 1688 based on estimation.

For LiLAC(Brakedown/Basefold) and Brakedown, we configured the code rate ρ to 1/2 and the relative distance δ to 0.09. Accordingly, we set the query count to $-128/\log_2(1 - 0.09/2) = 1927$ for Brakedown implementations.

Analysis. We compare field-agnostic MLPCS: LiLAC(Brakedown), LiLAC(Basefold), Brakedown, and Basefold, as shown in Table 3. Our scheme applies the Spielman code for reductions from $N = 2^{30}$ to $2^{30} \to 2^{28} \to 2^{26}$ and from $N = 2^{28}$ to $2^{28} \to 2^{26}$, followed by Brakedown or Basefold MLPCS at $N = 2^{26}$. As shown in Figure 3, LiLAC incurs the same costs as Brakedown/Basefold from $N = 2^{22}$ to 2^{26} due to the absence of reduction.

In terms of proving time, LiLAC(Basefold) demonstrates significantly better performance compared to Basefold. For example, at an input size of 2^{28}, LiLAC(Basefold) completes in approximately 671 seconds, while Basefold requires 1034 seconds, making LiLAC(Basefold) around 1.5× faster. For larger input sizes, although proving times are not available for Basefold, LiLAC(Basefold) continues to offer competitive proving times compared to other schemes, making it highly suitable for efficiently handling large datasets. In contrast, LiLAC(Brakedown) shows proving times similar to Brakedown.

For proof size at an input size of 2^{30}, LiLAC(Brakedown) achieves a compact size of about 36MB, which is approximately 3.7× smaller than Brakedown's 133MB. Similarly, at an input size of 2^{28}, LiLAC(Basefold) offers a proof size of 29MB, around 1.4× smaller than Basefold's 40MB. This reduction in proof size helps reduce storage costs, and both protocols are particularly effective when minimizing proof size is critical.

Regarding verifier time at an input size of 2^{30}, LiLAC(Basefold) provides extremely fast verification, taking just 0.19 seconds compared to Brakedown's 3.8 seconds, making it about 20× faster. LiLAC(Brakedown) takes around 1.7 seconds, which is still about twice as fast as Brakedown. LiLAC(Basefold) offers the fastest verification performance overall.

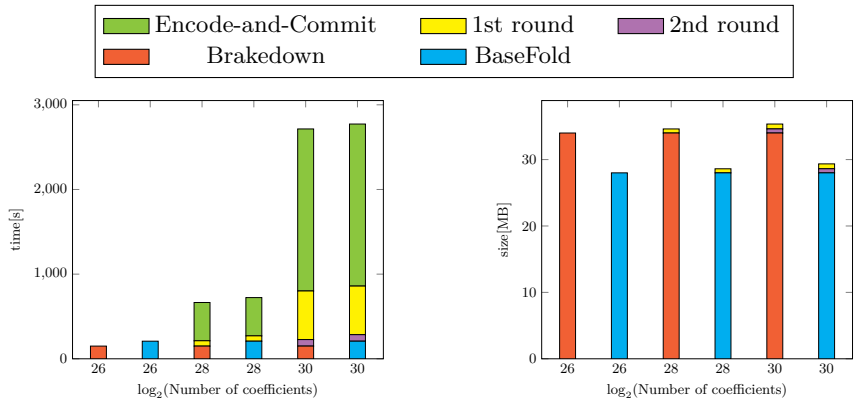
Table 3: Field-agnostic Multilinear Polynomial Commitment Schemes Comparison with single core over 128-bit field and 128-bit security.

	2^{22}	2^{24}	2^{26}	2^{28}	2^{30}
Prover time(s)					
LiLAC(Brakedown)	9.09	33.15	150.41	613.27	2653.79
Brakedown				599.96	2605.95
LiLAC(Basefold)	12.53	49.94	208.14	671.00	2711.51
Basefold				1033.79	N/A
Proof size(MB)					
LiLAC(Brakedown)	9.60	17.83	34.44	35.05	35.79
Brakedown				67.37	133.27
LiLAC(Basefold)	13.98	19.76	28.57	29.18	29.92
Basefold				40.24	N/A
Verifier time(s)					
LiLAC(Brakedown)	0.40	0.41	1.67	1.69	1.71
Brakedown				1.78	3.80
LiLAC(Basefold)	0.08	0.12	0.15	0.17	0.19
Basefold				0.31	N/A

In summary, LiLAC(Basefold) excels in verification speed and proof size, making it especially well-suited for applications that require rapid verification and minimal proof size. On the other hand, LiLAC(Brakedown) achieves linear proving time and maintains proving times comparable to Brakedown, while outperforming LiLAC(Basefold) with slightly faster proving times. However, it introduces minor overhead in proof size and verification time compared to LiLAC(Basefold), making it a well-balanced option for scenarios that prioritize quick proving times while still benefiting from reduced proof sizes and faster verification compared to Brakedown.

Costs at Each Step. We analyze how effective the reduction is in our scheme. We divide the process of the scheme into Encode-and-Commit, reduction, and MLPCS, and examine how the reduction contributes to the overall time and size. We analyze the effectiveness of the reductions by comparing the proving time and proof size of our LiLAC using Spielman codes across different coefficient sizes, as shown in Figure 5.

Figure 5(a) provides a detailed view of the proving time for LiLAC(Brakedown) and LiLAC(Basefold) when using Spielman codes. Notably, the Encode-and-Commit phase accounts for the majority of each bar. This indicates that most of the time is spent during the initial phase of Eval, where the polynomial with the largest coefficient is encoded and committed. In the case of Spielman codes, when the coefficient size is 2^{30} or 2^{28}, the reduction process ultimately reduces it to 2^{26}. Thanks to this reduction process, the stages that reduce both 2^{30} and 2^{28} to 2^{26} show similar proving times, suggesting that when reductions lead to the same intermediate coefficient size, the proving times become comparable.



(a) Proving time with Spielman code (b) Proof size with Spielman code

Fig. 5: Cost for each round

Figure 5(b) shows the proof size when using Spielman codes. In most bars, Brakedown and Basefold account for the majority of the proof size, indicating that these two protocols generate relatively large proofs. In contrast, each time LiLAC-Reduce is applied to reduce the polynomial's coefficient size, the proof size decreases, resulting in a relatively compact overall proof size. When using Spielman codes, the proof size generated by LiLAC-Reduce is relatively small compared to the total proof size, demonstrating that our LiLAC effectively manages proof size even for large coefficient sizes.

This analysis reveals that while the Encode-and-Commit phase accounts for the majority of the proving time, the MLPCS used plays a critical role in determining proof size. In particular, the proving time is proportional to the initial coefficient size, and excluding the Encode-and-Commit phase, the proving time in other stages decreases sharply. This highlights the significant role of the reduction technique in enhancing efficiency when handling polynomials with large coefficient sizes and underscores the importance of developing efficient encoding methods to reduce the time spent in the Encode-and-Commit phase.

4.2 Field-specific PCS comparison

Parameters. We evaluate the performance of field-specific polynomial commitment schemes over 128-bit fields, setting the security parameter $\lambda = 100$ for all protocols. For BasefoldFri, we followed the analysis of Reed-Solomon codes in [GLS+23, ACFY24a, ACFY24b], setting the relative distance δ to $(1-\rho)/2$ (unique decoding). With a code rate of $\rho = 1/2$, the query count for BasefoldFri is calculated as $-100/\log_2(1-1/4) = 241$. For Orion, we configure the code rate $\rho = 1/2$ and the relative distance $\delta = 0.09$. Accordingly, the query count is set to $-100/\log_2(1-0.09/2) = 1506$ to implement Orion. For FRI, STIR, WHIR,

and LiLAC(Shockwave), we configure the code rate ρ and ℓ based on the capacity bound (in this configuration, $\delta := 1 - \rho - \eta$) defined in WHIR [ACFY24b].

Additionally, to achieve a round-by-round soundness error of 2^{-100}, a constant proof-of-work with 22 bits is uniformly applied for all sizes N [ACFY24a, ACFY24b] in FRI, STIR, WHIR, and LiLAC(Shockwave). Proof-of-work is a method used in the Fiat-Shamir heuristic to transform an interactive protocol into a non-interactive one, by requiring multiple hash computations to generate randomness instead of a single computation. Specifically, it involves finding an n such that $\mathsf{H}(n, \tau) < 2^{|\mathsf{H}|}/D$, where τ represents a context, $|\mathsf{H}|$ denotes the output bit size of the hash, and D indicates the difficulty. By adjusting the difficulty D, the number of hash computations required to find n can be controlled. For example, with $D = 2^{22}$, an average of 2^{22} hash computations are needed. In this case, the number of hash operations required for an attacker to succeed with a probability of 2^{-t} is 2^{t+22}. Hence, for an attacker to succeed after 2^{λ} attempts, the attacker's success probability must be $2^{-\lambda} \cdot D$. When $D = 2^{22}$, the IOP soundness error can be relaxed from 2^{-100} to 2^{-78} while maintaining the same hash computation effort required for the attacker to succeed.

Analysis. We compare our LiLAC(Shockwave) with field-specific MLPCSs, such as BasefoldFri, Orion, and WHIR, and field-specific univariate PCSs, such as FRI and STIR, as shown in Table 4. By applying the RS code, we significantly reduce the proof size and verification time, but this comes at the cost of losing the field-agnostic property.

As shown in Table 4, LiLAC(Shockwave) demonstrates outstanding performance across all input sizes N compared to competing schemes in terms of proving time, proof size, and verification time.

For proving time, LiLAC(Shockwave) performs similarly to or slightly faster than WHIR and STIR for smaller input sizes, while clearly outperforming BasefoldFri and Orion. As the input size increases, the performance gap becomes even more pronounced. In large-scale input environments such as 2^{28} and 2^{30}, LiLAC(Shockwave) significantly outpaces all schemes. For instance, at 2^{30}, LiLAC (Shockwave) is 4.7× faster than Orion, 2.8× faster than WHIR, and 2.3× faster than STIR.

In terms of proof size, LiLAC(Shockwave) is also highly efficient. At 2^{28}, it achieves a proof size 74× smaller than BasefoldFri. At 2^{30}, the proof size is about 52× smaller than Orion and 27% smaller compared to WHIR and STIR.

For verification time, LiLAC(Shockwave) outperforms all other schemes. At 2^{28}, LiLAC(Shockwave) achieves a verification time of 1.4ms, which is about 30× faster than BasefoldFri and 68× faster than Orion. It also outperforms WHIR and STIR, consistently showing better performance across all input sizes. At 2^{30}, LiLAC(Shockwave) provides verification times that are 38% faster than STIR and 14% faster than WHIR.

In conclusion, LiLAC(Shockwave) maintains balanced performance across all input sizes N, while the performance gap with competing schemes becomes more pronounced in large-scale data environments. Specifically, compared to existing MLPCS's, LiLAC(Shockwave) demonstrates superior performance in all as-

Table 4: Field-specific Polynomial Commitment Schemes Comparison with a single core over 128-bit field and 100-bit security. Note that FRI, STIR, and WHIR only verify the proximity test. In the case of LiLAC, for $N = 2^{28}, 2^{30}$, each round is set with $-\log \rho = (1, 4, 6, 8, 3)$, while for $N = 2^{22}, 2^{24}, 2^{26}$, $-\log \rho$ is set to $(1, 2, 4, 6, 8, 3)$ where ρ denotes the code rate.

N	2^{22}	2^{24}	2^{26}	2^{28}	2^{30}
Prover time(s)					
LiLAC(Shockwave)	8.8	19.0	65.6	251.8	1145.9
BasefoldFri	12.0	48.3	202.2	927.6	N/A
Orion	10.5	47.3	206.8	905.2	5449.6
WHIR	10.3	37.9	165.5	738.7	3268.8
FRI	6.2	25.9	106.7	443.3	1922.8
STIR	8.5	36.4	144.6	617.9	2610.9
Proof size(KB)					
LiLAC(Shockwave)	91	96	102	90	96
BasefoldFri	4274	5027	5840	6710	N/A
Orion	3942	4193	4457	4744	5027
WHIR	91	100	111	121	132
FRI	174	210	249	293	338
STIR	90	101	110	122	131
Verifier time(ms)					
LiLAC(Shockwave)	1.4	1.5	1.6	1.4	1.8
BasefoldFri	24	26	30	43	N/A
Orion	66	72	81	96	157
WHIR	1.5	1.8	2.1	1.9	2.1
FRI	2.7	3.3	4.0	4.6	5.3
STIR	2.2	2.4	2.5	2.8	2.9

pects: proving time, proof size, and verification time. These characteristics make LiLAC(Shockwave) the optimal choice for practical use, whether in small-scale data or large-scale data environments.

Round-by-Round Performance. Table 5 illustrates the reduction process for 2^{30}. The size of N decreases progressively to 2^{30}, 2^{24}, 2^{18}, 2^{14}, and 2^{12}. For each size of N, the code rate ρ is set to 1/2, 1/16, 1/64, 1/256, and 1/8, respectively. After all reduction steps, we used Shockwave. Each reduction process includes the encoding and committing required for the next round, so the encode and commit needed in Shockwave were set with $\rho = 1/8$ and calculated in $\mathsf{rn} = 4$. Additionally, the testing parameter ℓ required in each round depends on the code rate of the previous round. For example, in $\mathsf{rn} = 1$, $\lceil 78/1 \rceil = 78$; in $\mathsf{rn} = 2$, $\lceil 78/4 \rceil = 20$; in $\mathsf{rn} = 3$, $\lceil 78/6 \rceil = 13$; and in $\mathsf{rn} = 4$, $\lceil 78/8 \rceil = 10$.

Our reduction process consists of 5 steps for 2^{22}, 2^{24}, and 2^{26}, and 4 steps for 2^{28} and 2^{30}. In each step, we use specific code rates ρ and testing parameters ℓ as follows:

Table 5: Round-by-round performance evaluation for LiLAC at $N = 2^{30}$.

LiLAC	Encoding + Committing + Evaluating	rn = 1	rn = 2	rn = 3	rn = 4	Shockwave	total
N	2^{30}	$2^{30} \to 2^{24}$	$2^{24} \to 2^{18}$	$2^{18} \to 2^{14}$	$2^{14} \to 2^{12}$	2^{12}	-
$-\log \rho$	1	4	6	8	3	-	-
ℓ	-	78	20	13	10	26	-
Proving time(s)	1025.03	117.27	5.55	1.25	0.01	0.0006	1149.12
Verifying time(ms)	-	0.93	0.20	0.12	0.08	0.52	1.85
Proof size(KB)	-	52.67	15.67	10.30	7.86	9.87	96.37
Verifier hashes	-	1684	500	328	25	212	2974

- For 2^{22}, 2^{24}, and 2^{26}: $\rho = 1/2$, in the encoding, committing, and evaluating; $\rho = 1/4$ and $\ell = 78$, in the first step; $\rho = 1/16$ and $\ell = 39$, in the second step; $\rho = 1/64$ and $\ell = 20$, in the third step; $\rho = 1/256$ and $\ell = 13$, in the fourth step; $\rho = 1/8$ and $\ell = 10$, in the fifth step.
- For 2^{28} and 2^{30}: $\rho = 1/2$, in the encoding, committing, and evaluating; $\rho = 1/16$ and $\ell = 78$, in the first step; $\rho = 1/64$ and $\ell = 20$, in the second step; $\rho = 1/256$ and $\ell = 13$, in the third step; $\rho = 1/8$ and $\ell = 10$, in the fourth step.

5 Conclusion

In this paper, we propose a field-agnostic, transparent multilinear polynomial commitment scheme. The proposed scheme theoretically achieves linear proving time, logarithmic verifying time, and logarithmic proof size with respect to the number of polynomial coefficients. This complexity surpasses that of any currently known code-based polynomial commitment schemes. Beyond theoretical efficiency, our scheme also demonstrates superior performance and proof size in practical implementation compared to existing multilinear polynomial commitment schemes. The proposed scheme employs a tensor IOPP-based approach, where the verification of tensor IOPP is proven via sum-check, and our scheme is recursively applied to verify the sum-check. This approach iteratively reduces a given multilinear polynomial to one with fewer coefficients; once the coefficients can no longer be reduced, existing multilinear polynomial schemes are applied to the final polynomial. Thus, the effectiveness of our scheme improves with the quality of the final multilinear polynomial scheme applied. In the future, we will develop multilinear polynomial commitment schemes that yield small proof size with reasonable performance for polynomials with a reduced number of coefficients.

References

ACFY24a. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. Stir: Reed-solomon proximity testing with fewer queries. In *Annual Interna-*

tional Cryptology Conference, pages 380–413. Springer, 2024.

ACFY24b. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. Whir: Reed–solomon proximity testing with super-fast verification. *Cryptology ePrint Archive*, 2024.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

Bas. Basefold. *Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes*.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

BCG20. Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020*. Springer, 2020.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *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 39*, pages 677–706. Springer, 2020.

BSBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *icalp 2018*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

BSCG+16. Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Short interactive oracle proofs with constant query complexity, via composition and sumcheck. *IACR Cryptol. ePrint Arch.*, 2016:324, 2016.

BSCR+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 38*, pages 103–128. Springer, 2019.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 499–530. Springer, 2023.

DP24. Benjamin E. Diamond and Jim Posen. Proximity testing with logarithmic randomness. *IACR Communications in Cryptology*, 1(1), 2024.

GLS+23. Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and field-agnostic snarks for r1cs. In *Annual International Cryptology Conference*, pages 193–226. Springer, 2023.

GWC19. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

KZG10. Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC 2021*. Springer, 2021.

LFKN92. Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

Ori. Orion. *Zero Knowledge Proof with Linear Prover Time*.

PHGR16. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

PST13. Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography Conference*, pages 222–242. Springer, 2013.

RRR16. Omer Reingold, Guy N Rothblum, and Ron D Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 49–62, 2016.

RS60. Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

RZR24. Noga Ron-Zewi and Ron Rothblum. Local proofs approaching the witness length. *Journal of the ACM*, 71(3):1–42, 2024.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

Spi95. Daniel A Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 388–397, 1995.

WTS+18. Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE, 2018.

XZS22. Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV*, pages 299–328. Springer, 2022.

ZCF23. Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. *Cryptology ePrint Archive*, 2023.

ZGK+17. Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vsql. *Cryptology ePrint Archive*, 2017.

ZGK+18. Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vram: Faster verifiable ram with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 908–925. IEEE, 2018.

ZXZS20. Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876. IEEE, 2020.

A Linear-time Multilinear Polynomial Commitments

In this appendix, we recall linear-time polynomial commitment schemes for multilinear polynomials. Our scheme, LiLAC, reduces the size of the coefficients through a reduction process before performing a polynomial commitment scheme to complete verification. We employ the PCS as a black-box component and enhance efficiency by using linear-time PCSs such as [GLS+23, XZS22].

These PCSs are advancements based on the work of Bootle et al. [BCG20]. In [BCG20], they define tensor IOP, a generalization of the interactive oracle proof (IOP), and design a PCS with a linear-time prover using a linear-time encodable code. The key technique for the linear prover is to reduce the encoding time by utilizing a linear-time encodable code, and the verifier checks whether the encoding was correctly computed by performing a proximity test on ℓ random entries rather than checking every entry.

Brakedown [GLS+23] expresses multilinear polynomials using Lagrange basis coefficients and commits to those coefficients using a Merkle tree hash function. To achieve a linear-time prover, it also uses an efficient linear-time encodable error-correcting code. Brakedown generalizes the results from [BCG20] and demonstrates the existence of PCS with the following complexity.

Theorem 4. *For security parameter λ and a positive integer t, given a hash function that can compute a Merkle-hash of N elements of $\mathbb{F}$ with the same time complexity as $\mathcal{O}(N)$ $\mathbb{F}$-ops, there exists a linear-time polynomial commitment scheme for multilinear polynomials. Specifically, there exists an algorithm that, given as input the coefficient vector of an r-variate multilinear polynomial over $\mathbb{F}$ over the Lagrange basis, with $N = 2^r$, commits to the polynomial, where:*

- *the size of the commitment is $\mathcal{O}_\lambda(1)$,*
- *the running time of the commit algorithm is $\mathcal{O}(N)$ operations over $\mathbb{F}$.*

Furthermore, there exists a non-interactive argument of knowledge in the random oracle model to prove the correct evaluation of a committed polynomial with the following parameters:

- *the prover's running time is $\mathcal{O}(N)$ operations over $\mathbb{F}$,*
- *the verifier's running time is $\mathcal{O}_\lambda(N^{1/t})$ operations over $\mathbb{F}$,*
- *the proof size is $\mathcal{O}_\lambda(N^{1/t})$.*

Brakedown specifically achieves $\mathcal{O}(N)$ prover time, $\mathcal{O}(\sqrt{N})$ verification time, and proof size for matrices, which are tensors of order 2. Additionally, one of the key features of Brakedown is that it is field-agnostic, meaning the scheme does not depend on the choice of field.

Orion [XZS22] improves upon Brakedown's results by utilizing a technique called code-switching [RZR24]. This code-switching method allows for efficient proof composition, maintaining the linear prover while reducing both the verification time and proof size to $\mathcal{O}(\log^2 N)$. Like Brakedown, Orion uses the generalized Spielman code, a linear-time encodable code, to achieve a linear prover. To efficiently locate this code, Orion introduces a testing algorithm for lossless expander graphs.

B Proof of Theorem 1

Proof. **Completeness:** Let $r = \log N$. If the round is the last, this follows trivially from the completeness of the linear-time polynomial commitment scheme. Now, let us assume that the round is not the last. For $0 \leq e \leq N-1$, define the binary vector $\mathbf{e} = (e_0, e_1, \ldots, e_{r-1})$, where $e = (e_{r-1} \cdots e_1 e_0)_2$ represents the binary expansion of e. Given that $\mathbf{C} \in \mathbb{F}^N$, we can interpret C as a function from $\{0,1\}^r$ to $\mathbb{F}$, defined by $C(\mathbf{e}) = \mathbf{C}[e]$. From the definition of the eq function, the multilinear extension of eq is given by $\widetilde{\mathsf{eq}}(\mathbf{x}, \mathbf{e}) := \prod_{i=1}^{r} ((1-e_i)(1-x_i) + e_i x_i)$. Thus, the polynomial $f_N^{\mathbf{C}}(x_1, \cdots, x_r)$ can be expressed as

$$f_N^{\mathbf{C}}(x_1, \cdots, x_r) = \sum_{\mathbf{e} \in \{0,1\}^r} \widetilde{\mathsf{eq}}(\mathbf{x}, \mathbf{e}) \cdot C(\mathbf{e}).$$

In our protocol, the same procedure is applied to the vector $\mathbf{z}$ for the consistency test as well as to $\mathbf{w}$ for the proximity test, so it is sufficient to prove completeness for $\mathbf{z}$ alone.

The value σ, which represents the evaluation of $f_N^{\mathbf{C}}(x_1, \cdots, x_r)$ at the vector $\mathbf{z}$, can be written as:

$$\sigma = f_N^{\mathbf{C}}(z_1, \cdots, z_r) = \sum_{\mathbf{e} \in \{0,1\}^r} \widetilde{\mathsf{eq}}(\mathbf{z}, \mathbf{e}) \cdot C(\mathbf{e}).$$

Using the definition of $\widetilde{\mathsf{eq}}$, the vectors $\mathbf{l_z}$ and $\mathbf{r_z}$ can be expressed as

$$\mathbf{l_z} = \otimes_{i=1}^{r/2}(1 - z_i, z_i) = \left(\widetilde{\mathsf{eq}}(\mathbf{z}_{[1..r/2]}, \mathbf{i})\right)_{\mathbf{i} \in \{0,1\}^{r/2}},$$

and

$$\mathbf{r_z} = \otimes_{i=r/2+1}^{r}(1 - z_i, z_i) = \left(\widetilde{\mathsf{eq}}(\mathbf{z}_{[r/2+1..r]}, \mathbf{i})\right)_{\mathbf{i} \in \{0,1\}^{r/2}}.$$

Using the vector $\mathbf{r_z}$, the vector $\mathbf{y_z}$ is computed by folding the matrix $\mathbf{C}$ as follows:

$$\mathsf{Fold}(\mathbf{C}; \mathbf{r_z}) := \mathbf{C} \cdot \mathbf{r_z} = \left(\sum_{j=1}^{k} \mathbf{C}[j, i] \cdot \widetilde{\mathrm{eq}}(\mathbf{z}_{[r/2+1..r]}, \mathbf{i}) \right)$$

where $\mathbf{i} \in \{0,1\}^{\frac{r}{2}}$. Finally, using the vector $\mathbf{l_z}$, the folding of the vector $\mathbf{y_z}$ is given by:

$$\mathsf{Fold}(\mathbf{y_z}; \mathbf{l_z}) = \left(\widetilde{\mathrm{eq}}(\mathbf{z}_{[1..r/2]}, \mathbf{j})\right)^T \cdot \left(\sum_{j=1}^{k} \mathbf{C}[j,i] \cdot \widetilde{\mathrm{eq}}(\mathbf{z}_{[r/2+1..r]}, \mathbf{i})\right).$$

where $\mathbf{j} \in \{0,1\}^{\frac{r}{2}}$ and $\mathbf{i} \in \{0,1\}^{\frac{r}{2}}$. This simplifies to:

$$\begin{aligned}
&\sum_{i=1}^{k}\sum_{j=1}^{k} \widetilde{\mathrm{eq}}(\mathbf{z}_{[1..r/2]}, \mathbf{j}) \cdot \mathbf{C}[j,i] \cdot \widetilde{\mathrm{eq}}(\mathbf{z}_{[r/2+1..r]}, \mathbf{i}) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{k} \widetilde{\mathrm{eq}}(\mathbf{z}_{[1..r/2]}, \mathbf{j}) \cdot \widetilde{\mathrm{eq}}(\mathbf{z}_{[r/2+1..r]}, \mathbf{i}) \cdot \mathbf{C}[j,i] \\
&= \sum_{\mathbf{e}\in\{0,1\}^r} \widetilde{\mathrm{eq}}(\mathbf{z}, \mathbf{e}) \cdot C(\mathbf{e}).
\end{aligned}$$

Hence, we conclude that $\sigma = f_N^{\mathbf{C}}(\mathbf{z}) = \mathbf{l}_\mathbf{z}^T \cdot \mathbf{C} \cdot \mathbf{r_z} = \mathbf{l}_\mathbf{z}^T \cdot \mathbf{y_z}$.

Since for every $l_t \in I$, $\mathsf{Enc}(\mathbf{y_z})_{l_t} = \mathsf{Fold}(\mathbf{E})_{l_t}$ (by Proposition 1), we obtain the following ℓ identities:

$$\sum_{i=1}^{k} \mathbf{G}[l_t, i] \cdot \mathbf{y_z}[i] = \sum_{i=1}^{k} \mathbf{E}[l_t, i] \cdot \mathbf{r_z}[i],$$

where $\mathbf{G}$ is the encoding matrix for Enc and $\mathbf{E} = \mathsf{Enc}(\mathbf{C})$. Each vector $\mathbf{G}[l_t, :]$, $\mathbf{y_z}$, $\mathbf{E}[l_t, :]$, and $\mathbf{r_z}$ has length k. Let the multilinear extensions of these vectors be denoted as $\widetilde{G}_t(\mathbf{x})$, $\widetilde{y}_\mathbf{z}(\mathbf{x})$, $\widetilde{E}_t(\mathbf{x})$, and $\widetilde{r}_\mathbf{z}(\mathbf{x}) \in \mathbb{F}[x_1, \ldots, x_{\log k}]$, respectively:

$$\begin{aligned}
\widetilde{G}_t(\mathbf{x}) &:= \sum_{\mathbf{e}\in\{0,1\}^{\log k}} \widetilde{eq}(\mathbf{x}, \mathbf{e}) \cdot G(\mathbf{l}_t || \mathbf{e}), \\
\widetilde{y}_\mathbf{z}(\mathbf{x}) &:= \sum_{\mathbf{e}\in\{0,1\}^{\log k}} \widetilde{eq}(\mathbf{x}, \mathbf{e}) \cdot y_\mathbf{z}(\mathbf{e}), \\
\widetilde{E}_t(\mathbf{x}) &:= \sum_{\mathbf{e}\in\{0,1\}^{\log k}} \widetilde{eq}(\mathbf{x}, \mathbf{e}) \cdot E(\mathbf{l}_t || \mathbf{e}), \\
\widetilde{r}_\mathbf{z}(\mathbf{x}) &:= \sum_{\mathbf{e}\in\{0,1\}^{\log k}} \widetilde{eq}(\mathbf{x}, \mathbf{e}) \cdot r_\mathbf{z}(\mathbf{e}).
\end{aligned}$$

where $\mathbf{l}_t$ is binary vector of l_t.

By definition, $N' = \psi(2(\ell+1)\sqrt{N})$, and since $\frac{N'}{2k} \geq \ell$, the ℓ multilinear polynomials $\widetilde{G}_t(\mathbf{x})$ (and $\widetilde{E}_t(\mathbf{x})$) can be represented as a single multivariate polynomial through multilinear extension using $\log \frac{N'}{2k}$ variables $y_1, \ldots, y_{\log \frac{N'}{2k}}$. This polynomial is defined as follows: $\widetilde{G}(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{e}\in\{0,1\}^{\log \frac{N'}{2k}}} \widetilde{\mathrm{eq}}(\mathbf{y}, \mathbf{e}) \cdot \widetilde{G}_e(\mathbf{x})$ $\Big($ and $\widetilde{E}(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{e}\in\{0,1\}^{\log \frac{N'}{2k}}} \widetilde{\mathrm{eq}}(\mathbf{y}, \mathbf{e}) \cdot \widetilde{E}_e(\mathbf{x})\Big)$. Therefore, the ℓ identities can

be combined into a single polynomial form, which satisfies the following property for all $\mathbf{j} \in \{0,1\}^{\log \frac{N'}{2k}}$:

$$\sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{G}(\mathbf{j},\mathbf{i})\widetilde{y}_{\mathbf{z}}(\mathbf{i}) = \sum_{\mathbf{i}\in\{0,1\}^{\log k}} \widetilde{E}(\mathbf{j},\mathbf{i})\widetilde{r}_{\mathbf{z}}(\mathbf{i}).$$

Defining $S_{\mathbf{z}}(\mathbf{y},\mathbf{x})$ as $S_{\mathbf{z}}(\mathbf{y},\mathbf{x}) := \widetilde{G}(\mathbf{y},\mathbf{x})\widetilde{y}_{\mathbf{z}}(\mathbf{x}) - \widetilde{E}(\mathbf{y},\mathbf{x})\widetilde{r}_{\mathbf{z}}(\mathbf{x})$, we see that for all $\mathbf{j} \in \{0,1\}^{\log \frac{N'}{2k}}$, the sum $\sum_{\mathbf{i}\in\{0,1\}^{\log k}} S_{\mathbf{z}}(\mathbf{j},\mathbf{i}) = 0$. This indicates that all ℓ identities hold true. Consequently, for any vector $\mathbf{c} \leftarrow\!\!\$\ \mathbb{F}^{\log \frac{N'}{2k}}$, we have $\sum_{\mathbf{i}\in\{0,1\}^{\log k}} S_{\mathbf{z}}(\mathbf{c},\mathbf{i}) = 0$. Similarly, since folding the vector $\mathbf{y}_{\mathbf{z}}$ with the vector $\mathbf{l}_{\mathbf{z}}$ yields $\mathsf{Fold}(\mathbf{y}_{\mathbf{z}};\mathbf{l}_{\mathbf{z}}) = \sigma$, we obtain the following identity: $\sum_{\mathbf{i}\in\{0,1\}^{\log k}} T(\mathbf{i}) = \sigma$, where $T(\mathbf{x}) = \widetilde{l}_{\mathbf{z}}(\mathbf{x}) \cdot \widetilde{y}_{\mathbf{z}}(\mathbf{x})$.

After performing the sum-check protocol, in order to compute the input for recursive rounds, we define:

- $N' := \psi(2(\ell+1)\sqrt{N})$
- $\mathbf{C}' := \Big(G[l_1,:],\cdots,G[l_\ell,:],\mathbf{0},\mathbf{y}_{\mathbf{z}},E[l_1,:],\cdots,E[l_\ell,:],\mathbf{0},\mathbf{y}_{\mathbf{w}}\Big) \in \mathbb{F}^{N'}$
- $\mathbf{z}' := (\mathbf{c}\,||\,\mathbf{b}\,||\,a) \in \mathbb{F}^{\log N'}$
- $f_{N'}^{\mathbf{C}'}(x_1,\cdots,x_{\log N'}) := \sum_{\mathbf{e}\in\{0,1\}^{\log N'}} \widetilde{\mathrm{eq}}(\mathbf{x},\mathbf{e}) \cdot C'(\mathbf{e})$

Then, the completeness of our protocol is established through the following steps:

$$\begin{aligned}
&f_{N'}^{C'}(\mathbf{z}') = f_{N'}^{C'}(\mathbf{c}||\mathbf{b}||a)\\
&= \sum \widetilde{eq}\big((\mathbf{c}||\mathbf{b}||a),(\mathbf{e_1}||\mathbf{e_2}||e_3)\big)\cdot C'(\mathbf{e_1}||\mathbf{e_2}||e_3)\\
&= \sum_{\mathbf{e_1}\in\{0,1\}^{\log\frac{N'}{2k}}} \widetilde{eq}(\mathbf{c},\mathbf{e_1})\Big(\sum_{\mathbf{e_2}\in\{0,1\}^{\log k}} \widetilde{eq}(\mathbf{b},\mathbf{e_2})\Big(\sum_{e_3\in\{0,1\}} \widetilde{eq}(a,e_3)\cdot C'(\mathbf{e_1}||\mathbf{e_2}||e_3)\Big)\Big)\\
&= \sum_{\mathbf{e_1}\in\{0,1\}^{\log\frac{N'}{2k}}} \widetilde{eq}(\mathbf{c},\mathbf{e_1})\Big(\Big(\widetilde{G_i}(\mathbf{b})+\widetilde{y}_{\mathbf{z}}(\mathbf{b})\Big)(1-a)+\Big(\widetilde{E_i}(\mathbf{b})a+\widetilde{y}_{\mathbf{w}}(\mathbf{b})\Big)a\Big)\\
&= (1-a)\left(\widetilde{G}(\mathbf{c},\mathbf{b})+\prod_{i=1}^{\log\frac{N'}{2k}} c_i\cdot\widetilde{y}_{\mathbf{z}}(\mathbf{b})\right)+a\left(\widetilde{E}(\mathbf{c},\mathbf{b})+\prod_{i=1}^{\log\frac{N'}{2k}} c_i\cdot\widetilde{y}_{\mathbf{w}}(\mathbf{b})\right)\\
&= \sigma'.
\end{aligned}$$

Soundness: The soundness error in the membership proof step, denoted by ϵ_{mp}, results from the sum of errors in each of the Merkle tree proofs. Each error corresponds to the probability of a collision occurring at each layer of the Merkle tree. Therefore, the total soundness error for the 2ℓ commitments cm_{E_t} and cm_{G_t} is $\epsilon_{mp} = \frac{2\ell\cdot\mathsf{depth}}{|\mathbb{F}|}$, where $\mathsf{depth} = \log n$. In our protocol, the verifier selects ℓ random rows to perform proximity and consistency tests. The soundness error for these tests, derived in [BCG20,GLS+23], is given as $\mathcal{O}\left(\frac{d^2}{|\mathbb{F}|} + \left(1-\frac{\delta}{3}\right)^{\ell}\right)$, particularly since we consider the case of $t = 2$. Additionally, based on the latest version

of [AHIV17][8], the second term $\left(1-\frac{\delta}{3}\right)^{\ell}$ can be optimized to $\left(1-\frac{\delta}{2}\right)^{\ell}$. Thus, the total error probability for these tests becomes $\epsilon_{test} = \mathcal{O}\left(\frac{d^2}{|\mathbb{F}|} + \left(1-\frac{\delta}{2}\right)^{\ell}\right)$. Let ϵ_{sc} represent the soundness error that arises within the sum-check protocol. The polynomials $T(\mathbf{x})$, $S_{\mathbf{z}}(\mathbf{y},\mathbf{x})$, and $S_{\mathbf{w}}(\mathbf{y},\mathbf{x})$ used in this protocol are polynomials in $\log k$ variables, where each variable has a maximum degree of 2. Thus, the soundness error for each sum-check protocol is $\frac{\log k}{|\mathbb{F}|}$, leading to a total of $\epsilon_{sc} = \frac{3\log k}{|\mathbb{F}|}$. The total soundness error is therefore given by

$$\begin{aligned}\epsilon_{\text{Reduce}} &= \epsilon_{mp} + \epsilon_{sc} + \epsilon_{test} \\ &= \mathcal{O}\left(\frac{\ell \cdot \log n + d^2 + \log k}{|\mathbb{F}|} + \left(1-\frac{\delta}{2}\right)^{\ell}\right)\end{aligned}$$

Thus, the LiLAC-Reduce protocol has the following soundness error depending on the value of the round:

$$\begin{cases}\epsilon_{\text{Reduce}} & \text{if round is not last} \\ \epsilon_{\text{MLPCS}} & \text{if round is last}\end{cases}$$

Here, ϵ_{MLPCS} represents the soundness error of the linear-time polynomial commitment employed by the prover and verifier when the round is last. Consequently, the overall soundness error of LiLAC-Reduce is given by $\epsilon_1 = \max\{\epsilon_{\text{Reduce}}, \epsilon_{\text{PCS}}\}$.

[8] We referenced https://eprint.iacr.org/2022/1608 and used the soundness analysis presented in Appendix C.