Veloz: Efficient and Flexible Distribution Framework for Interleaved-Code-Based Polynomial Commitment Scheme

Yuanzhuo Yu[1], Shi-Feng Sun[1,*], Yuncong Zhang[2,*], Chenhua Fan[1,†], Tianyi Ma[1], Dawu Gu[1,*]
[1]*Shanghai Jiao Tong University* [2]*Shandong University*
{*yzyu2000, shifeng.sun, mty021129, dwgu*}*@sjtu.edu.cn, yuncong@sdu.edu.cn, ord1r3hv@gmail.com*

***Abstract*—Polynomial commitment schemes (PCSs) are a fundamental cryptographic primitive that allows a prover to reveal evaluations for a committed polynomial. Motivated by the inefficiency of proof generation for large-scale computations as well as the concerns regarding third-party reliance and quantum threats, a line of recent works has focused on distributing code-based PCS, where the proving workload is distributed among multiple sub-provers to accelerate proof generation, while preserving transparent setup and plausible quantum resilience. However, for M sub-provers generating an evaluation proof for a polynomial of size N, existing solutions either require $O(N)$ total communication among sub-provers, or incur an $O(M)$ overhead in proof size.**

In this paper, we introduce Veloz, a novel distribution framework for interleaved-code-based multilinear PCS, which *for the first time* achieves communication cost sublinear in N, and eliminates the dependence of proof size on the number of sub-provers, without compromising proving speed or security. At its core is a customized proof aggregation method from interleaved code that efficiently combines sub-proofs via minimum communication. We further present two instantiations of Veloz: one based on Reed-Solomon code, Veloz_{RS}, achieves $O(\frac{N}{M}\log\frac{N}{\log N})$ proving time, $O(\lambda\cdot\frac{\log^2 N}{\log\log N}+M\cdot\frac{N}{\log N})$ communication, and $O(\lambda\cdot\frac{\log^2 N}{\log\log N})$ proof size; the other based on the fast linear code from Brakedown (Golovnev et al., CRYPTO 2023), $\text{Veloz}_{\text{Lin}}$, features $O(\frac{N}{M})$ proving time, $O(\lambda\cdot K+M\cdot\frac{N}{K})$ communication, and $O(\lambda\cdot K)$ proof size for $K\in[\sqrt[3]{N},\sqrt{N}]$, while enjoying field agnosticism.

We also implement both schemes in Rust and conduct a comprehensive performance evaluation. The experimental results demonstrate their efficient scalability with increasing M. More specifically, for $N=2^{30}$ and $M=8$, Veloz_{RS} takes 61.8s for proof generation, achieving a 6.46× speedup compared to running a single prover, while $\text{Veloz}_{\text{Lin}}$ generates a proof in 26.9s and achieves a 7.03× speedup.

1. Introduction

A *polynomial commitment scheme* (PCS) [1] enables a prover to commit to a polynomial f with degree bound d for each variable, and later to convince a verifier that f satisfies the degree bound and that $f(x)=y$ for some point x and value y, by generating a proof. Since the seminal work by Kate et al. [1], a large body of research has been conducted over the last two decades [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. Generally, the existing works can be classified into two categories: pairing-based PCSs [1], [2], [3], which enjoy fast verification, small proof size, and efficient proof aggregation, due to the algebraic homomorphisms rooted in bilinear groups; and code-based PCSs [4], [5], [6], [7], [8], [9], [10], [11], which in contrast feature a transparent setup and plausible post-quantum security, as they rely on collision resistant hash functions.

In recent years, PCS has become a crucial building block of *succinct non-interactive argument of knowledge* (SNARK) [16], [17], [18], [19], [20], [21], [22], [23], which enables a powerful prover to efficiently convince a resource-constrained verifier of the truth of a public statement. SNARK's capacity to reconcile the inherent tradeoff between efficiency and security has driven its adoption in cryptocurrencies [24], blockchain scaling [25], and machine learning inference [26]. With the continuously growing scales of computation, however, the slow proof generation of PCS has been widely recognized as a main bottleneck of SNARK, in terms of both proving time and memory usage. To circumvent these issues, intensive studies [12], [13], [14], [15], [27], [28], [29], [30], [31], [32] have resorted to distributed computation, which distributes the proving workload across multiple sub-provers that run in parallel and cooperate via communication. Nevertheless, the primary challenges in designing such distributed schemes lie in *accelerating proof generation, minimizing sub-prover communication, and distributing memory consumption, while preserving low verification cost and compact proof size*.

In order to achieve high verification efficiency and tiny proof size, most schemes [27], [28], [29], [30], [31], [32] start with the pairing-based PCSs, the homomorphism property of which not only enables the sub-provers to generate sub-proofs locally with minimum communication, but also allows a master sub-prover to aggregate them with no extra overhead. For a polynomial of size N and M sub-provers, the state-of-the-art pairing-based distributed PCSs from HyperPianist [30] and Cirrus [31] achieve $O(\frac{N}{M})$ proving complexity, $O(M\log\frac{N}{M})$ communication among sub-provers, and $O(\log\frac{N}{M})$ verification cost and proof size.

∗. Corresponding authors.
†. Part of this work was completed while Fan was at Xidian University.

TABLE 1: Comparisons of Veloz to prior code-based distributed PCSs. Here N is the size of polynomial, M the number of sub-provers, and $T := \frac{N}{M}$. K is a flexible parameter, where $K \in [1, T]$ in FRIttata, and $K \in [\sqrt[3]{N}, \sqrt{N}]$ is a power of two in $\mathsf{Veloz}_{\mathsf{Lin}}$. "Field-agnostic?" means whether PCS works over any sufficiently large finite field $\mathbb{F}$.

Schemes	Proving	Total communication	Proof size	Verification	Field-agnostic?
deVirgo [12]	$O(T \log T)$	$O(N)$	$O(\lambda \cdot M + \log^2 T)$	$O(\lambda \cdot M + \log^2 T)$	No
FRIttata [13]	$O(T \log T)^\star$	$O(MK + \lambda \cdot M \log \frac{T}{K} \log T)$	$O\Big(\lambda\big(M \log \frac{T}{K} \log T + \log^2 K\big)\Big)$	$O\Big(\lambda\big(M \log \frac{T}{K} \log T + \log^2 K\big)\Big)$	No
HyperFond [14]	$O(T \log T)$	$O(\lambda \cdot M \log^2 T)$	$O(\lambda \cdot M \log^2 T)$	$O(\lambda \cdot M \log^2 T)$	Yes
$\mathsf{DePIP}_{\mathsf{FRI}}$ [15]	$O(T \log T)$	$O(N)$	$O(\lambda \cdot M + \log^2 T)$	$O(\lambda \cdot M + \log^2 T)$	No
$\mathsf{Veloz}_{\mathsf{RS}}$ (**ours**)	$O(T \log \frac{N}{\log N})$	$O(\lambda \cdot \frac{\log^2 N}{\log \log N} + M \cdot \frac{N}{\log N})$	$O(\lambda \cdot \frac{\log^2 N}{\log \log N})$	$O(\lambda \cdot \frac{\log^2 N}{\log \log N})$	No
$\mathsf{Veloz}_{\mathsf{Lin}}$ (**ours**)	$O(T)$	$O(\lambda \cdot K + M \cdot \frac{N}{K})$	$O(\lambda \cdot K)$	$O(\lambda \cdot (K + \frac{N}{K}))$	Yes

* The proving time in FRIttata is $O(MK + M \log \frac{T}{K} \log T + \log^2 K)$, assuming each sub-prover already holds the evaluations of its polynomial [13, Sec. 4], which originates from an outer protocol therein. We view it as an independent PCS in this table.

However, most pairing-based distributed PCSs suffer from the need for a trusted third party to generate public parameters during setup. Moreover, the underlying hardness assumptions therein are vulnerable to quantum algorithms [33]. Motivated by these concerns, a line of recent works [12], [13], [14], [15] focuses on distributing code-based PCSs, as they feature transparent setup and plausible post-quantum security, as well as implementation-friendliness in engineering. Due to the lack of homomorphic properties over bilinear groups, however, existing code-based distributed PCSs either impose an $O(M)$ overhead in proof size and verification cost, or require $O(N)$ total communication among sub-provers, constrained by a seemingly inevitable tradeoff between efficient proving and fast verification. Based on the discussions above, a natural research question that remains open is:

How to design a transparent and plausibly quantum secure distributed PCS featuring communication sublinear in the size of polynomial, and compact proof size independent of sub-prover number?

In this paper, we make affirmative progress on this question. Based on Ligerito [34], a recent multilinear PCS with interleaved-code structure, we introduce Veloz, a distribution framework for interleaved-code-based multilinear PCS that separates the distribution layer from the choice of the underlying linear code, and instantiate it with Reed-Solomon (RS) code and the fast linear code from Brakedown [8], respectively. Both are transparent and plausibly post-quantum secure; the generated proofs have sizes independent of M, requiring communication cost sublinear in N.

1.1. Our Contributions

We put forward Veloz, a novel distribution framework for interleaved-code-based multilinear PCS, and instantiate it with RS and Brakedown codes, respectively. In contrast to previous code-based works, both schemes enjoy a transparent setup, plausible quantum resilience, communication sublinear in polynomial size, and sub-prover number independent proof; comparisons are shown in Table 1. In more detail, our main contributions are summarized as follows.

We propose the *first* distribution framework, dubbed Veloz, that resolves the tradeoff between communication and proof size. To achieve communication sublinear in the polynomial size while keeping proof size independent of the number of sub-provers, our key abstraction is to decouple the distribution mechanisms from row-wise encoding, allowing the underlying efficient linear code to be instantiated independently. To realize this abstraction, we distribute the partial SumCheck in Ligerito, and develop an interleaved-code-specific aggregation method for Merkle commitment and linear combination. Jointly, these techniques avoid both $O(N)$ communication and an $O(M)$ overhead on proof size.

Then we instantiate Veloz with RS code and the linear-time encodable code in Brakedown [8], and obtain two concretely efficient schemes, $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$. Regarding $\mathsf{Veloz}_{\mathsf{RS}}$, it achieves $O(\frac{N}{M} \log \frac{N}{\log N})$ proving time, $O(\lambda \cdot \frac{\log^2 N}{\log \log N} + M \cdot \frac{N}{\log N})$ communication, and $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ proof size[1]; also it features efficient verification with complexity $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$, which applies even in the single-prover setting. As to $\mathsf{Veloz}_{\mathsf{Lin}}$, it achieves *for the first time* linear proving time, benefiting from Brakedown's linear code, and $O(\lambda \cdot K + M \cdot \frac{N}{K})$ communication and $O(\lambda \cdot K)$ proof size for $K \in [\sqrt[3]{N}, \sqrt{N}]$. Moreover, it enjoys *field agnosticism* as Brakedown, enabling application to any sufficiently large finite field without fast fourier transform (FFT)-friendly structure required by RS code-based schemes.

Finally, we implement both $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$ in Rust, and conduct a comprehensive performance evaluation. We simulate up to 16 sub-provers on a single cloud server, and run experiments for multilinear polynomials with different numbers of variables. The experimental results exhibit a linear improvement in proving efficiency as the number of sub-provers increases. More specifically, for $N = 2^{30}$ and $M = 8$, $\mathsf{Veloz}_{\mathsf{RS}}$ generates a proof with 61.8s, achieving a 6.46× speedup over the single-prover baseline, with a proof size of 554KB, and verification time of 4.23ms. Under the same settings, $\mathsf{Veloz}_{\mathsf{Lin}}$ performs proof generation in 26.9s and achieves a 7.03× speedup; its proof length is 49.2MB, with 1.04s for verification. We also compare to FRIttata [13], HyperFond [14] and $\mathsf{DePIP}_{\mathsf{FRI}}$ [15] based on their publicly available source codes in the same environment,

1. As analyzed in Sec. 4 and App. C, $\mathsf{Veloz}_{\mathsf{RS}}$ achieves the optimal proof size $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$, conditioned on $M \leq O(\log N)$. We note here that our general design is not subjected to this constraint, where M is only bounded by the row number of the encoded matrix.

where we fix $M = 8$. The results demonstrate that both $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$ are faster and more memory efficient for proof generation. In particular, given $N = 2^{30}$, HyperFond and FRIttata already run out of memory, while $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$ are 1.20× and 2.76× faster than $\mathsf{DePIP}_{\mathsf{FRI}}$, respectively. We provide more details in Sec. 5.

Beyond the distributed construction, we also give the formal security proofs of binding and extractability properties in Sec. 4, which applies to the single-prover setting as well, thus implying binding and extractability of [34]. Recall that [34] only shows the completeness and soundness analysis. In addition, we provide a more succinct verification algorithm in Sec. 3.4 with $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ complexity, improving over the $O(N)$ verification of [34].

1.2. Overview

In this section, we first briefly review the background of code-based PCS and prior distribution schemes with tradeoffs between linear communication and $O(M)$ overhead in proof size, to better understand our motivation. Then, by identifying the structural requirements for resolving this tradeoff, we abstract a distribution interface, and progressively examine several constructions exhibiting such structural patterns and their limitations. Finally, we analyze the challenges in extending these constructions to distributed settings, and unfold our main ideas for overcoming them.

Motivation. Often, code-based PCSs build upon a special proof system known as *Interactive Oracle Proof of Proximity* (IOPP) [5], [9], [35], [36], [37], where a prover $\mathcal{P}$ first Merkle commits to a vector $\mathbf{u}$ over a finite field $\mathbb{F}$, and sends $\mathsf{MC}(\mathbf{u})$ to a verifier $\mathcal{V}$; $\mathcal{V}$ then tests that $\mathbf{u}$ is close to some prescribed linear code space $\mathcal{C}$, by querying $\mathcal{P}$ and verifying $\Theta(\lambda)$ Merkle paths through $\mathbf{u}$ to $\mathsf{MC}(\mathbf{u})$, where λ is the security parameter. In the PCS setting, given a polynomial with coefficient or evaluation vector $\mathbf{f}$, $\mathcal{P}$ argues that it satisfies the degree bound by first encoding $\mathbf{f}$, and then invoking an IOPP to prove that the distance of the encoded vector $\hat{\mathbf{f}} = \mathsf{Enc}(\mathbf{f})$ from $\mathcal{C}$ is small.

When it comes to distributed schemes [12], [13], [14], [15], the vector $\mathbf{f}$ is evenly distributed to M sub-provers $\{\mathcal{P}_i\}_{i=0}^{M-1}$ for local commitment, with $\mathcal{P}_i$ holding $\mathsf{MC}(\hat{\mathbf{f}}^{(i)})$, and the distance claim for $\hat{\mathbf{f}}$ from $\mathcal{C}$ is reduced to M claims for $\{\hat{\mathbf{f}}^{(i)}\}_{i=0}^{M-1}$ from $\hat{\mathcal{C}}$, a sub-code space with adapted dimension. Now, we arrive at the following dilemma: to prove that $\hat{\mathbf{f}}^{(i)}$ is close to $\hat{\mathcal{C}}$ for each i, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ either (1) fully exchange these sub-vectors to respond to $\mathcal{V}$'s query for Merkle paths of $\mathsf{MC}(\hat{\mathbf{f}})$, which renders M-independent proof size but incurs $O(N)$ communication [12], [15], or (2) let $\mathcal{V}$ query the Merkle paths of $\mathsf{MC}(\hat{\mathbf{f}}^{(i)})$ for each i, which reduces communication among sub-provers, while producing an M-fold proof size [14]. FRIttata [13] is a combination of these strategies, by "flexibilizing" the moment when sub-provers exchange elements, to obtain a balance between communication and proof size tuned by a parameter $K \in [1, \frac{N}{M}]$. However, its experimental results still report asymptotical linear growth of proof size as M increases.

We observe that the $O(M)$ overhead in [13], [14] essentially stems from hashings during Merkle commitment, because they spoil existing *inner* algebraic relations between $\{\hat{\mathbf{f}}^{(i)}\}_{i=0}^{M-1}$ and $\hat{\mathbf{f}}$. Consequently, unless $O(|\mathbf{f}|) = O(N)$ communication is allowed, efficient aggregation and alignment for Merkle paths of $\{\hat{\mathbf{f}}^{(i)}\}_{i=0}^{M-1}$ is hindered, forcing $\mathcal{V}$ to treat each as an independent codeword. Therefore, the inner relations between encoded sub-vectors and the original codeword should be avoided in our design.

Intuitively, the dilemma could be resolved if computing $\hat{\mathbf{f}} = \mathsf{Enc}(\mathbf{f})$ already requires evenly splitting $\mathbf{f}$ into multiple slices for parallel encoding, where each encoded slice is no longer related to others. Because with such decoupling, their encoding can be treated in a black-box way and evenly distributed to sub-provers. After this encoding-independent distribution, it suffices to focus on the *outer* design among these codewords, i.e., how we arrange their Merkle commitments and linear combinations with communication sublinear in N. Motivated by this observation, we abstract the following structural requirements for distribution-friendly design.

Distribution interface. We identify two structural properties that enable distribution-friendliness for a code-based PCS: (1) the encoding of a polynomial admits a partition into locally processable components, so encoding and calculation can be performed without relying on algebraic relations among different components, and (2) the proximity test supports efficient aggregation across these components, rather than requiring each one to be tested independently. The former separates the distribution of the encoding layer from the commitment structure, while the latter enables sub-provers to jointly produce a response to the verifier's query.

This interface suggests that a feasible distribution should simultaneously enable independent encoding and processing of polynomial components, and support proof aggregation across their encoded forms. While such a combination may seem difficult to achieve, the following interleaved-code-based IOPP exhibits these structural properties, and provides a promising starting point to substantially improve the tradeoff between communication and proof size.

• **Ligero [4] and logarithmic randomness [38].** Ligero, proposed by Ames et al., introduces an IOPP that tests multiple vectors in one shot with *interleaved linear code*. Roughly speaking, given a linear code $\mathcal{C}$ with block length n, one can collect n' codewords of $\mathcal{C}$ as a $\mathbb{F}^{n' \times n}$-matrix $\hat{\mathbf{U}}$, and view it as a codeword of the interleaved linear code $\mathcal{C}^{n'}$, where each row $\hat{\mathbf{U}}[i, \cdot]$ is a codeword in $\mathcal{C}$. To test the membership of $\hat{\mathbf{U}}$ in $\mathcal{C}^{n'}$, the verifier $\mathcal{V}$ obtains $\mathsf{MC}(\hat{\mathbf{U}})$ from the prover $\mathcal{P}$, challenges $\mathcal{P}$ with a linear randomness $\mathbf{r} \leftarrow_{\$} \mathbb{F}^{n'}$, and receives a vector $\hat{\mathbf{u}} \in \mathbb{F}^n$ claimed to be $\mathbf{r} \cdot \hat{\mathbf{U}}$. $\mathcal{V}$ then queries $\Theta(\lambda)$ columns of $\hat{\mathbf{U}}$, and, for each queried $\hat{\mathbf{U}}[\cdot, j]$, verifies if $\langle \mathbf{r}, \hat{\mathbf{U}}[\cdot, j] \rangle = \hat{\mathbf{u}}[j]$. Following this pattern, Diamond and Posen [38] replace the linear randomness $\mathbf{r}$ with one generated by *logarithmic randomness*, i.e., first sample $\mathbf{r}' \leftarrow_{\$} \mathbb{F}^{\log n'}$ and then derive the continuous tensor product $\mathbf{r}_{\otimes} = (1 - \mathbf{r}'[1], \mathbf{r}'[1]) \otimes \cdots \otimes (1 - \mathbf{r}'[\log n'], \mathbf{r}'[\log n']) \in \mathbb{F}^{n'}$.

This IOPP is a potential solution to our research question, because if we arrange the encoded coefficient vector

into a $n' \times n$ matrix $\hat{\mathbf{U}} = \mathbf{Mat}_{n' \times n}(\hat{\mathbf{f}})$ with row-major fashion, and assign each sub-prover $\frac{n'}{M}$ rows of $\hat{\mathbf{U}}$, then the sub-columns distributed among $\{\mathcal{P}_i\}_{i=0}^{M-1}$ are naturally *aligned*, where the only requirement is $M \leq n'$. With this customized distribution method for interleaved code-based IOPP, it takes just $O(\lambda \cdot n')$ communication to aggregate queried columns, and $O(M \cdot n)$ for linear combination, both below $O(n' \cdot n) = O(N)$. During the process, proof size stabilizes, since distribution does not affect how each row $\mathbf{U}[i, \cdot]$ is encoded or linearly combined.

However, the proof size of Ligero-based IOPP and PCS [8], [37] amounts to $O(\lambda \cdot n' + n) = \Omega(\sqrt{\lambda \cdot N})$, as opposed to $O(\lambda \cdot M \log^2 \frac{N}{M})$, raising an obstacle for practical use in large-scale computation. Therefore, before distribution, we should seek other methods that shrink proof length without sacrificing the previous IOPP. Fortunately, the following work presents a possible solution.

• **Ligerito [34].** Recently, Novakovic and Angeris provide a lightweight code-based PCS for multilinear polynomials, by iteratively combining Ligero IOPP with logarithmic randomness. The choice of matrix size $n' \times n$ therein is not fixed, which offers much flexibility for proof size; in the best case, the derived proof size is only $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$. We now recall their main ideas.

Given a μ-variate multilinear polynomial $f \in \mathcal{F}_\mu^{(\leq 1)} := \mathbb{F}^{(\leq 1)}[x_1, \cdots, x_\mu]$, a value $y \in \mathbb{F}$, and an evaluation point $\mathbf{z} \in \mathbb{F}^\mu$, the claim $y = f(\mathbf{z})$ can be expressed as $y = \sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) \cdot z(\mathbf{b}) = \langle \mathbf{u}, \mathbf{z}_\otimes \rangle$, where $B_\mu := \{0,1\}^\mu$, z is the polynomial interpolation of the vector $\mathbf{z}_\otimes := \otimes_{i=1}^{\mu} (1 - \mathbf{z}[i], \mathbf{z}[i]) \in \mathbb{F}^{2^\mu}$, and $\mathbf{u} := f(B_\mu) = (f(0 \cdots 00), f(0 \cdots 01), \cdots, f(1 \cdots 11)) \in \mathbb{F}^{2^\mu}$ collects sequential evaluations. The protocol first "partially" invokes SumCheck for $\mu' < \mu$ rounds, reducing $y = \langle \mathbf{u}, \mathbf{z}_\otimes \rangle$ to $y' = \sum_{\mathbf{b} \in B_{\mu-\mu'}} f(\mathbf{r}, \mathbf{b}) \cdot z(\mathbf{r}, \mathbf{b}) = \langle \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{u}), \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{z}_\otimes) \rangle = \langle \mathbf{u}', \mathbf{w} \rangle$, where $\mathbf{r} \in \mathbb{F}^{\mu'}$ is the generated randomness, and $\mathbf{r}_\otimes := \otimes_{i=1}^{\mu'} (1 - \mathbf{r}[i], \mathbf{r}[i])$.

To check this claim, it suffices for $\mathcal{V}$ to acquire $\mathbf{u}'$, as $\mathbf{r}_\otimes$ and $\mathbf{z}$ are known to $\mathcal{V}$, hence so is $\mathbf{w}$. To show the validity of $\mathbf{u}'$, both sides engage in a Ligero IOPP with logarithmic randomness $\mathbf{r}_\otimes$, which proves the vector-matrix multiplication $\mathbf{u}' = \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{u})$. In more detail, let $\mathcal{P}$ encode $\mathbf{U} := \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{u})$ row-wise with a public generator matrix $\mathbf{G}$, Merkle commit to $\hat{\mathbf{U}} := \mathbf{U} \cdot \mathbf{G}$, and send $\mathsf{MC}(\hat{\mathbf{U}})$ to $\mathcal{V}$. $\mathcal{V}$ then queries for $\Theta(\lambda) =: |I|$ columns of $\hat{\mathbf{U}}$ indexed by a set I, dubbed $\hat{\mathbf{U}}_I$. After verifying $\hat{\mathbf{U}}_I$ with $\mathsf{MC}(\hat{\mathbf{U}})$ and the corresponding Merkle paths, $\mathcal{V}$ checks $\mathbf{r}_\otimes \cdot \hat{\mathbf{U}}_I = \mathbf{u}' \cdot \hat{\mathbf{G}}_I$, where $\hat{\mathbf{G}}_I$ are $\mathbf{G}$'s indexed columns with the same index set I. Then, $\mathcal{V}$ accepts $\mathbf{u}'$, and locally verifies $y' = \langle \mathbf{u}', \mathbf{w} \rangle$.

The authors of Ligerito notice the following room for iteration: the verification for $\mathbf{r}_\otimes \cdot \hat{\mathbf{U}}_I = \mathbf{u}' \cdot \hat{\mathbf{G}}_I$ and $y' = \langle \mathbf{u}', \mathbf{w} \rangle$ are essentially $|I| + 1$ inner products regarding $\mathbf{u}'$, which aligns with the evaluations of $f(\mathbf{r}, \cdot) \in \mathcal{F}_{\mu-\mu'}^{(\leq 1)}$ over $B_{\mu-\mu'}$. Hence, verification can be reduced to

$$\begin{aligned} \big\langle (\mathbf{r}_\otimes \cdot \hat{\mathbf{U}}_I) \| y', \mathbf{s} \big\rangle =: y'' &= \langle \mathbf{u}', \mathbf{z}' \rangle \\ &:= \big\langle f(\mathbf{r}, B_{\mu-\mu'}), (\mathbf{G}_I \| \mathbf{w}^\top) \cdot \mathbf{s} \big\rangle, \end{aligned}$$

where $\mathbf{s} \in \mathbb{F}^{|I|+1}$ is randomness for batching. The new inner product has a similar form as $y = \langle \mathbf{u}, \mathbf{z}_\otimes \rangle$, yet with a $2^{-\mu'}$-fold scale. Following this, one can iteratively reduce the inner product interleaved with an intermittent SumCheck for f, until the scale is tiny enough for a direct verification.

Challenges ahead. Together, Ligero and Ligerito suggest that the distribution interface above provides a viable method towards improving the tradeoff between communication and proof size: Ligero enables aggregation across distributed components, while Ligerito further shrinks the proof through iteration. However, the iterative structure introduces new dependencies that complicate distribution. Specifically, distributing Ligerito requires handling both the partial SumCheck and the vector-matrix multiplication within the iterative Ligero-based IOPPs.

Strategies for SumCheck distribution are anything but new. As introduced in [12] and applied in [14], [30], [31], the key is to number $M = 2^m$ sub-provers with binary representation of indices, i.e., $\mathbf{bin}(i) := (b_{m-1}, \cdots, b_0)$ for $\mathcal{P}_i$, where $i = \sum_{j=0}^{m-1} 2^j b_j$. Then, $\sum_{\mathbf{b} \in B_\mu} f(\mathbf{b})$ can be decomposed as $\sum_{i=0}^{M-1} \sum_{\mathbf{b}' \in B_{\mu-m}} f(\mathbf{b}', \mathbf{bin}(i))$, and each $\mathcal{P}_i$ processes $f^{(i)} := \sum_{\mathbf{b}' \in B_{\mu-m}} f(\mathbf{b}', \mathbf{bin}(i))$ locally, with the master sub-prover $\mathcal{P}_0$ delegated to complete the final m rounds of interaction with $\mathcal{V}$. However, its direct application to Ligerito heavily burdens $\mathcal{P}_0$, as partial SumCheck is essentially a standard one with an intermittent pattern, producing multiple such "final m rounds".

Additionally, the Ligero IOPP in each iteration requires a single commitment to the row-encoded matrix and a row-wise linear combination. A naive distribution would either gather the encoded matrix at one sub-prover, causing redundant communication, or maintain M independent Merkle trees, reintroducing an $O(M)$ overhead on proof size. Thus, preserving a single global commitment with minimum communication is the main challenge in aggregation.

Beyond the challenges of distribution, the $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ proof size comes at the cost of $O(N)$ verification, as directly calculating $\mathbf{w} = \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{z}_\otimes)$ requires $O(2^\mu) = O(N)$ field operations. Therefore, verification also demands further simplification to achieve succinctness.

Our construction. Now, we present our ideas on addressing these challenges, yielding a distribution framework for multilinear PCS within the aforementioned distribution interface, with communication sublinear in N and proof size independent of M.

• **Distributed partial SumCheck.** In each iteration, a μ'-round partial SumCheck is invoked to reduce the claim $y = \sum_{\mathbf{b} \in B_\mu} f(\mathbf{b})$ to another one $y' = \sum_{\mathbf{b} \in B_{\mu-\mu'}} f(\mathbf{r}, \mathbf{b})$, where $\mathbf{r} \in \mathbb{F}^{\mu'}$ is the generated randomness. A direct application of the distribution method [12] above mandates that $\mathcal{P}_0$ parse $\mathbf{r} = \mathbf{r}_1 \| \mathbf{r}_2 \in \mathbb{F}^{(\mu'-m)+m}$, with only $\mu' - m$ rounds actually distributed, and the rest m rounds completed on $\mathcal{P}_0$'s own. Its burden surges as the number of iteration increases.

Our solution, instead of viewing the summation as a whole, is to distribute the workload in each round, regardless of the choice of μ'. The observation is, in round $1 \leq k \leq \mu'$,

$\{\mathcal{P}_i\}_{i=0}^{M-1}$ are computing $s_k(x_k) = \sum_{\mathbf{b}\in B_{\mu-k}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b})$ for $\mathcal{V}$, which is a binary tree-like summation. The initial $\mu-k-m$ levels of summation can be distributed among all sub-provers, until they get stuck, with each holding $s_k^{(i)}(x_k) = \sum_{\mathbf{b}'\in B_{\mu-k-m}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b}', \mathbf{bin}(i))$. To proceed, we *buffer* the rest m levels of summation: for $j = 1, \cdots, m$, for each $i < 2^{m-j}$, $\mathcal{P}_{i+2^{m-j}}$ sends $s_k^{(i+2^{m-j})}(x_k)$ to $\mathcal{P}_i$; then, each $\mathcal{P}_i$ updates $s_k^{(i)}(x_k) \leftarrow s_k^{(i)}(x_k) + s_k^{(i+2^{m-j})}(x_k)$. Such buffering halves the number of polynomials in each level, until finally $\mathcal{P}_0$ obtains $s_k(x_k) = s_k^{(0)}(x_k)$. As a result, $\mathcal{P}_0$ only processes $\log M$ polynomials in each iteration, instead of $O(M)$. Besides, this method requires *no* extra communication, because it only alters the destination each $s_k^{(i)}(x_k)$ is sent to. For detailed elaboration, please refer to Sec. 3.1.

• **Distributed vector-matrix multiplication.** Parallel to the partial SumCheck, in each iteration, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ also convince $\mathcal{V}$ that $\mathbf{u} = \mathbf{r}_\otimes \cdot \mathbf{U}$ with $\mathbf{U} \in \mathbb{F}^{2^{\mu'}\times 2^{\mu-\mu'}}$, by invoking an IOPP for the row-encoded matrix $\hat{\mathbf{U}}$. As sketched earlier, we leverage the proof aggregation method specialized from interleaved code, by assigning each $\mathcal{P}_i$ $2^{\mu'-m}$ rows of $\mathbf{U}$, denoted as $\mathbf{U}^{(i)} = \mathbf{U}[i\cdot 2^{\mu'-m} : (i+1)\cdot 2^{\mu'-m}, \cdot]$, for local encoding, Merkle commitment, and row-wise linear combination. While encoding can be viewed in a black-box way and instantiated with any efficient linear code, the other two processes do require careful arrangement, to minimize proof size and communication.

To Merkle commit to $\hat{\mathbf{U}}$ in the single-prover setting, $\mathcal{P}$ hashes $\hat{\mathbf{U}}$ column-wise, obtains a vector of hash values $\mathbf{h}$, and builds a Merkle tree with $\mathbf{h}$ being leaves. For distribution, after encoding $\mathbf{U}^{(i)}$ for $\hat{\mathbf{U}}^{(i)} = \mathbf{U}^{(i)} \cdot \mathbf{G}$, we let each $\mathcal{P}_i$ hash $\hat{\mathbf{U}}^{(i)}$ column-wise, and obtain $\mathbf{h}^{(i)}$. At this point, directly letting $\{\mathcal{P}_i\}_{i=0}^{M-1}$ compute M Merkle trees would result in an M-fold proof size. To avoid this overhead, we evenly partition and allocate $\{\mathbf{h}^{(i)}\}_{i=0}^{M-1}$. That is, $\mathcal{P}_i$ evenly partitions $\mathbf{h}^{(i)}$ into M slices, and sends the j-th slice to $\mathcal{P}_j$, $\forall j \neq i$. Now, each $\mathcal{P}_j$ holds several "semi-hashed" columns of $\hat{\mathbf{U}}$, and can proceed to complete column-wise hashing.

Linear combination follows a similar procedure, i.e., each $\mathcal{P}_i$ obtains $\mathbf{u}^{(i)} = \mathbf{r}_\otimes^{(i)} \cdot \mathbf{U}^{(i)} \in \mathbb{F}^{2^{\mu-\mu'}}$, and sends its j-th slice to $\mathcal{P}_j$ for local addition. An immediate advantage is, since $\mathbf{u} \in \mathbb{F}^{2^{\mu-\mu'}}$ will be reshaped into $\mathbf{U}' \in \mathbb{F}^{2^{\mu''}\times 2^{\mu-\mu'-\mu''}}$ for the next iteration, this cross-sending pattern provides each $\mathcal{P}_i$ with the target rows $\mathbf{U}'[i\cdot 2^{\mu''-m} : (i+1)\cdot 2^{\mu''-m}, \cdot]$ *for free*. It saves communication cost by a factor of 2, compared to delegating $\mathcal{P}_0$ alone for gathering $\{\mathbf{u}^{(i)}\}_{i=0}^{M-1}$ and broadcasting their sum to other sub-provers.

We provide detailed explanation for these three components in Sec. 3.2, and combine them with distributed partial SumCheck to obtain Veloz in Sec. 3.3.

• **Achieving an efficient verifier.** While distribution benefits prover side, verification is linear in N, from calculating $\mathbf{w} = \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'}\times 2^{\mu-\mu'}}(\mathbf{z}_\otimes)$. Towards this, authors of [34] point out the following tensor structure: $\mathbf{Mat}_{2^{\mu'}\times 2^{\mu-\mu'}}(\mathbf{z}_\otimes) = \otimes_{i=1}^{\mu'}(1-\mathbf{z}[i], \mathbf{z}[i])^\top \cdot \otimes_{i=\mu'+1}^{\mu}(1-\mathbf{z}[i], \mathbf{z}[i])$. Based on an equation $\langle \otimes_{i=1}^k \mathbf{u}_i, \otimes_{i=1}^k \mathbf{v}_i\rangle = \prod_{i=1}^k \langle \mathbf{u}_i, \mathbf{v}_i\rangle$ (where $|\mathbf{u}_i| = |\mathbf{v}_i|$ for each i), the verifier can succinctly compute $\mathbf{r}_\otimes \cdot \otimes_{i=1}^{\mu'}(1-\mathbf{z}[i], \mathbf{z}[i])^\top$. Nonetheless, verification still costs $O(\lambda\cdot\frac{N\log\log N}{\log^2 N})$, if we desire the $O(\lambda\cdot\frac{\log^2 N}{\log\log N})$ proof size. It does not leave much room for further optimization, because in the next iteration, $\mathbf{z}'$ is a random linear combination of the columns of $\mathbf{G}_I$ and $\mathbf{w}^\top$, losing tensor structure.

That said, we observe that things could be different if not only $\mathbf{w}$, but also columns $\mathbf{G}_I$ have tensor structures, because a finite tensor product $\mathbf{u}_\otimes = \otimes_{i<\infty}(1-\mathbf{u}[i], \mathbf{u}[i])$ can be split at any slot, for any number of slices, so the tensor structure preserved in $\mathbf{r}_\otimes \cdot \mathbf{Mat}(\mathbf{u}_\otimes)$ admits multiple left-multiplications from $\mathbf{r}'_\otimes\cdot$ in the upcoming iterations. Therefore, before combining columns $\mathbf{G}_I$ and $\mathbf{w}$ with $\mathbf{s}$ and translating $\mathbf{z}' = (\mathbf{G}_I \| \mathbf{w}^\top)\cdot\mathbf{s}$ to a new matrix $\mathbf{Mat}_{2^{\mu''}\times 2^{\mu-\mu'-\mu''}}(\mathbf{z}')$, we can swap the sequence of operators $\mathbf{Mat}(\cdot)$ and $\cdot\mathbf{s}$, and left-multiply $\mathbf{r}_\otimes$ to each column of $(\mathbf{G}_I\|\mathbf{w}^\top)$, to maximally exploit their tensor structures. Since generator matrix of RS code features this property (we can turn $\mathbf{G}$ into a *Vandermonde matrix* by row-wise linear transform), our intuition can be realized. For elaboration, please refer to Sec. 3.4 and Sec. 4.

1.3. Related Works

Pairing-based PCSs. Kate et al. [1] first propose the concept of PCS, and present one construction for univariate polynomials based on bilinear pairing. Papamanthou et al. [2] later generalize it to multivariate cases. Both schemes enjoy efficient verification (up to logarithmic), but they rely on a trusted setup, and require a common reference string (CRS) of linear size. Lee [3] improves both by transparentizing setup, and sub-linearizing CRS.

Code-based PCSs. IOPPs for constructing code-based PCSs can be roughly categorized into RS code-based and interleaved code-based. *Fast Reed-Solomon IOPP* (FRI) [5] pioneers the former category, followed by subsequent DEEP-FRI [6], STIR [10] and WHIR [11], which improve soundness analysis for better verification efficiency and smaller proof size. BaseFold [9] generalizes RS code-based IOPPs to *foldable* code-based ones, and features field agnosticism. Compared with FRI, BaseFold verifier makes more queries to lower soundness error. Its soundness is improved in DeepFold [39], which shortens proof length. PCSs built upon RS-code often enjoy up to polylogarithmic verification and proof size. The main drawback, however, is the quasilinear proving complexity, due to the invocation of FFTs.

On the other hand, Ligero [4] first studies interleaved code-based IOPPs. The idea is then refined by Bootle et al. [37] and in Brakedown [8]. When instantiated with a linear-time encodable code, interleaved code-based PCSs feature linear proving complexity, but with square-root verification and proof size. Based on this, Blaze [40] improves Brakedown on binary fields using the code-switching technique [41], which obtains polylogarithmic proof size.

Distributed PCSs. DIZK [27] first explores the distributed PCS and its application in distributed SNARK. Subsequent

works include Pianist [28], Hekaton [29], HyperPianist [30], Cirrus [31] and Soloist [32]. All constructions are built on [1] and [2] for verification efficiency and proof aggregation, benefiting from homomorphisms in bilinear groups. As an exception, HyperPianist also distributes [3], and is the only one among above with a transparent setup.

Another line of works distributes code-based PCSs. deVirgo [12] first considers their distribution, and proposes a solution for FRI [5]-based multilinear PCS. Recently, Li et al. introduce $\text{DePIP}_{\text{FRI}}$ [15], which improves deVirgo by allowing splitting the workload of a single polynomial across sub-provers. Both schemes yield polylogarithmic proof size with linear communication. Concurrent to [15], FRIttata [13] distributes bivariate polynomials built on FRI, and HyperFond [14] distributes BaseFold [9] multilinear PCS; both require only polylogarithmic communication, at the cost of an $O(M)$ overhead in proof size. All aforementioned code-based distributed PCSs enjoy a transparent setup and plausible quantum resilience.

Concurrent work. Very recently, Guo et al. propose LigeSIS [42], a distributed PCS with $O(\sqrt{\lambda \cdot MN})$ communication and $O(\lambda \cdot \log^2 N)$ proof size and verification. Compared to LigeSIS, Veloz targets a more general distribution framework agnostic to the underlying linear code, whereas LigeSIS is RS-code-based. Moreover, LigeSIS achieves lower communication than Veloz_{RS} by leveraging a precomputed lookup table, which trades $32\times$ of memory usage [42, Sec. 3.2] for communication efficiency.

2. Preliminaries

Notations. Let λ denote the security parameter. We write $f(\lambda) = \mathsf{poly}(\lambda)$ if $f(\lambda) = O(\lambda^c)$ for some $c \in \mathbb{N}$, and $f(\lambda) = \mathsf{negl}(\lambda)$ if $f(\lambda) = o(\lambda^{-c})$ for every $c \in \mathbb{N}$. An event occurs with *overwhelming probability* if it holds with probability $1 - \mathsf{negl}(\lambda)$. A *relation* $\mathcal{R}$ is a set of instance-witness pairs $(\mathbb{x}; \mathbb{w})$.

Let $\mathbb{F}$ be a finite field with $\log|\mathbb{F}| = \Omega(\lambda)$, and $\mathcal{F}_\mu^{(\leq d)} := \mathbb{F}^{(\leq d)}[x_1, \cdots, x_\mu]$ the set of μ-variate polynomials with $\deg(x_i) \leq d$ for each i. We write $r \leftarrow_\$ S$ for uniform sampling from a finite set S, and $[i : j] := \{i, i+1, \cdots, j-1\}$ for $i < j$. Lowercase boldface letters denote vectors, e.g., $\mathbf{u} = (\mathbf{u}[1], \cdots, \mathbf{u}[\mu]) \in \mathbb{F}^\mu$, and uppercase boldface letters denote matrices. For a vector $\mathbf{u} \in \mathbb{F}^{\mu\nu}$, $\mathbf{Mat}_{\mu\times\nu}(\mathbf{u})$ denotes the matrix from its row-major reshape. For a matrix $\mathbf{U}$, $\mathbf{U}[i : j, \cdot]$ denotes rows i through $j-1$, and for an index set $I = \{i_1, \cdots, i_{|I|}\}$, $\mathbf{U}_I$ denotes the submatrix formed by columns $i_1, \cdots, i_{|I|}$.

Tensor product. Given matrices $\mathbf{U} \in \mathbb{F}^{\mu\times\mu'}$ and $\mathbf{V} \in \mathbb{F}^{\nu\times\nu'}$, their *tensor product* $\mathbf{U} \otimes \mathbf{V}$ is defined as $(\mathbf{U}[i, j] \cdot \mathbf{V}) \in \mathbb{F}^{\mu\nu\times\mu'\nu'}$. For a vector $\mathbf{r} = (\mathbf{r}[1], \cdots, \mathbf{r}[\mu]) \in \mathbb{F}^\mu$, denote by $\mathbf{r}_\otimes \in \mathbb{F}^{2^\mu}$ the tensor product $(1 - \mathbf{r}[1], \mathbf{r}[1]) \otimes \cdots \otimes (1 - \mathbf{r}[\mu], \mathbf{r}[\mu]) = \otimes_{i=1}^\mu (1 - \mathbf{r}[i], \mathbf{r}[i])$.

Multilinear polynomial. For every map $f : B_\mu \to \mathbb{F}$, there exists a unique polynomial $\tilde{f} \in \mathcal{F}_\mu^{(\leq 1)}$ such that $\tilde{f}(\mathbf{b}) = f(\mathbf{b}), \forall \mathbf{b} \in B_\mu$. $\tilde{f}$ is called the *multilinear extension* (MLE) of f, and can be expressed as $\tilde{f}(x_1, \cdots, x_\mu) = \sum_{\mathbf{b}\in B_\mu} f(\mathbf{b}) \cdot \tilde{eq}_\mathbf{b}(x_1, \cdots, x_\mu)$, where $\tilde{eq}_\mathbf{b}(x_1, \cdots, x_\mu) = \prod_{i=1}^\mu (\mathbf{b}[i] \cdot x_i + (1 - \mathbf{b}[i]) \cdot (1 - x_i))$ is the MLE of $eq_\mathbf{b} : B_\mu \to \{0, 1\}$ that maps $\mathbf{b}$ to 1 and others to 0.

Multivariate SumCheck protocol [43]. In general, a multivariate SumCheck protocol enables a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that, with high probability, the sum $\sum_{\mathbf{b}\in B_\mu} f(\mathbf{b})$ equals a purported value v, where $f \in \mathcal{F}_\mu^{(\leq d)}$. In more detail, $\mathcal{P}$ and $\mathcal{V}$ engage in a μ-round reduction as follows.

In round $1 \leq k \leq \mu - 1$, $\mathcal{P}$ computes and sends $\mathcal{V}$ a univariate polynomial $s_k(x_k) := \sum_{\mathbf{b}\in B_{\mu-k}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b})$. Then, $\mathcal{V}$ verifies if $\deg(s_k) \leq d$ and $s_k(0) + s_k(1) = s_{k-1}(\mathbf{r}[k-1])$ (or $s_1(0) + s_1(1) = v$ for $k = 1$). If verification passes, $\mathcal{V}$ samples and sends $\mathbf{r}[k] \leftarrow_\$ \mathbb{F}$ to $\mathcal{P}$. The claim $\sum_{\mathbf{b}\in B_{\mu-k+1}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], \mathbf{b}) = s_{k-1}(\mathbf{r}[k-1])$ is then reduced to $\sum_{\mathbf{b}\in B_{\mu-k}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k], \mathbf{b}) = s_k(\mathbf{r}[k])$. Finally, in round $k = \mu$, after verification $\mathcal{V}$ samples $\mathbf{r}[\mu] \leftarrow_\$ \mathbb{F}$, and checks whether $f(\mathbf{r}[1], \cdots, \mathbf{r}[\mu]) = s_\mu(\mathbf{r}[\mu])$. If all verifications pass, $\mathcal{V}$ accepts.

Linear codes [38]. A linear error-correcting code with message length k and block length n is an injective map from $\mathbb{F}^k$ to a linear subspace $\mathcal{C} \subseteq \mathbb{F}^n$. $\mathcal{C}$ is associated with a generator matrix $\mathbf{G} \in \mathbb{F}^{k\times n}$, of which the rows form a basis of $\mathcal{C}$, and the encoding of $\mathbf{m} \in \mathbb{F}^k$ is computed as $\mathbf{m} \cdot \mathbf{G}$. The *code rate* of $\mathcal{C}$ is defined as $\rho := \frac{k}{n} \leq 1$, and the *minimum Hamming distance* of $\mathcal{C}$ is the minimum number of different entries between any $\mathbf{c}_1 \neq \mathbf{c}_2 \in \mathcal{C}$. If $\mathcal{C}$ has a minimum Hamming distance $d \in [0 : n + 1]$, we call $\mathcal{C}$ a $[n, k, d]$-*code*, and denote by $\delta_\mathcal{C} := \frac{d}{n}$ the *relative minimum distance* of $\mathcal{C}$. Specifically, if $d = n - k + 1$, then $\mathcal{C}$ is *maximum distance separable* (MDS).

Given a $[n, k, d]$-code $\mathcal{C}$ and an integer $m \geq 1$, the corresponding m-*fold interleaved code* is defined as $\mathcal{C}^m \subseteq (\mathbb{F}^n)^m$. Viewing $\mathcal{C}^m$ as matrices in $\mathbb{F}^{m\times n}$, we say that two such matrices $\mathbf{M}_1 \neq \mathbf{M}_2$ differ at a column if they differ at any entry of that column. A matrix $\mathbf{M} \in \mathbb{F}^{m\times n}$ is within distance d from $\mathcal{C}^m$ if $\exists \mathbf{M}^* \in \mathcal{C}^m$ such that $\mathbf{M}$ and $\mathbf{M}^*$ differ at up to d columns. Denote by $\Delta_\mathcal{C}^m(\mathbf{M}, \mathbf{M}^*)$ the set of column indices that $\mathbf{M}$ and $\mathbf{M}^*$ differ.

2.1. Mathematical Facts

Next, we review some mathematical facts that will be frequently used throughout the work.

Tensor product and inner product. Given three vectors $\mathbf{u} \in \mathbb{F}^\mu, \mathbf{v} \in \mathbb{F}^\nu$, and $\mathbf{w} \in \mathbb{F}^{\mu\nu}$, directly computing the inner product $\langle \mathbf{u} \otimes \mathbf{v}, \mathbf{w} \rangle$ costs $2\mu\nu$ multiplications. However, we can evenly partition $\mathbf{w}$ into μ slices $(\mathbf{w}_1 \| \cdots \| \mathbf{w}_\mu) \in (\mathbb{F}^\nu)^\mu$, so $\langle \mathbf{u} \otimes \mathbf{v}, \mathbf{w} \rangle = \sum_{i=1}^\mu \mathbf{u}[i] \cdot \langle \mathbf{v}, \mathbf{w}_i \rangle = \sum_{i=1}^\mu \langle \mathbf{u}[i] \cdot \mathbf{w}_i, \mathbf{v} \rangle = \mathbf{u} \cdot \mathbf{Mat}_{\mu\times\nu}(\mathbf{w}) \cdot \mathbf{v}^\intercal$, with $\mu\nu + \min(\mu, \nu)$ multiplications.

Another observation is, given $\{\mathbf{u}_i, \mathbf{v}_i \in \mathbb{F}^{\mu_i}\}_{i=1}^k$, we have $\langle \mathbf{u}_1 \otimes \mathbf{u}_2, \mathbf{v}_1 \otimes \mathbf{v}_2 \rangle = \langle \mathbf{u}_1[1] \cdot \mathbf{u}_2 \| \cdots \| \mathbf{u}_1[\mu_1] \cdot \mathbf{u}_2, \mathbf{v}_1[1] \cdot \mathbf{v}_2 \| \cdots \| \mathbf{v}_1[\mu_1] \cdot \mathbf{v}_2 \rangle = \langle \mathbf{u}_1, \mathbf{v}_1 \rangle \cdot \langle \mathbf{u}_2, \mathbf{v}_2 \rangle$, which recursively yields $\langle \otimes_{i=1}^k \mathbf{u}_i, \otimes_{i=1}^k \mathbf{v}_i \rangle = \prod_{i=1}^k \langle \mathbf{u}_i, \mathbf{v}_i \rangle$. The letter method requires only $\sum_{i=1}^k \mu_i + k - 1$ multiplications, compared to $3 \cdot \prod_{i=1}^k \mu_i$ for direct computation.

Multilinear polynomials, matrices and vectors. Given $f \in \mathcal{F}_\mu^{(\leq 1)}$ and $\mathbf{r} \in \mathbb{F}^\mu$, by linearity it holds that

$$f(\mathbf{r}) = (1 - \mathbf{r}[i]) \cdot f(\mathbf{r}[1], \cdots, \mathbf{r}[i-1], 0, \mathbf{r}[i+1], \cdots, \mathbf{r}[\mu]) + \mathbf{r}[i] \cdot f(\mathbf{r}[1], \cdots, \mathbf{r}[i-1], 1, \mathbf{r}[i+1], \cdots, \mathbf{r}[\mu])$$

for any $i \in \{1, \cdots, \mu\}$. After unfolding the above equation for all i, $f(\mathbf{r})$ can be rewritten as the inner product of two $\mathbb{F}$-vectors with dimension 2^μ, i.e., $f(\mathbf{r}) = \langle \mathbf{f}, \mathbf{r}_\otimes \rangle$, where $\mathbf{f} = f(B_\mu) = (f(00\cdots00), f(00\cdots01) \cdots, f(11\cdots11))$. With this observation, any vector $\mathbf{f} \in \mathbb{F}^{2^\mu}$ can be viewed as a multilinear polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$, and vice versa. Moreover, if we reshape $\mathbf{f}$ into a matrix $\mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{f})$ for any $\mu' \in \{1, \cdots, \mu\}$, then $f(\mathbf{r}) = \otimes_{i=1}^{\mu'}(1 - \mathbf{r}[i], \mathbf{r}[i]) \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{f}) \cdot \otimes_{i=\mu'+1}^{\mu}(1 - \mathbf{r}[i], \mathbf{r}[i])^\top$. That is, computing $\langle \mathbf{f}, \mathbf{r}_\otimes \rangle$ is equivalent to splitting $\mathbf{r}_\otimes$ into two parts, with the left and right each multiplying a properly rearranged matrix, respectively.

2.2. Polynomial Commitment Schemes

Definition 1 (Polynomial Commitment Scheme [9])**.** *A multilinear polynomial commitment scheme* PCS *consists of polynomial time algorithms* (Setup, Commit, Open, Eval)*:*

$\mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda, \mu)$*: on input the security parameter* λ *and the number of variables* $\mu \in \mathbb{N}$*, outputs public parameters* gp*.*

$(\mathsf{cm}, t) \leftarrow \mathsf{Commit}(\mathsf{gp}, f)$*: on input public parameters* gp *and a multilinear polynomial* f*, outputs a commitment* cm *and an opening hint* t*.*

$b \leftarrow \mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f, t)$*: on input public parameters* gp*, a commitment* cm*, a multilinear polynomial* f *and an opening hint* t*, outputs a bit* b *for acceptance or rejection.*

$b \leftarrow \mathsf{Eval}(\mathsf{gp}, \mathsf{cm}, \mathbf{z}, y; f, t)$*: this is a protocol between a prover* $\mathcal{P}$ *and a verifier* $\mathcal{V}$ *with public parameters* gp*, a commitment* cm*, an evaluation point* $\mathbf{z} \in \mathbb{F}^\mu$ *and a value* $y \in \mathbb{F}$*.* $\mathcal{P}$ *additionally knows a multilinear polynomial* f *and an opening hint* t*, and tries to convince* $\mathcal{V}$ *that* $\mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f, t) = 1$ *and that* $f(\mathbf{z}) = y$*. At the end of the protocol,* $\mathcal{V}$ *outputs a bit* b *denoting acceptance or rejection.*

PCS *is complete if for any* $f \in \mathcal{F}_\mu^{(\leq 1)}$ *and* $\mathbf{z} \in \mathbb{F}^\mu$*,*

$$\Pr\left[\begin{array}{c} \mathsf{Eval}(\mathsf{gp}, \mathsf{cm}, \mathbf{z}, \\ f(\mathbf{z}); f, t) = 1 \end{array} \middle| \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda, \mu) \\ (\mathsf{cm}, t) \leftarrow \mathsf{Commit}(\mathsf{gp}, f) \end{array} \right] = 1.$$

PCS *is binding if for any* $\mu \in \mathbb{N}$ *and PPT adversary* $\mathcal{A}$*, the following probability is* $\mathsf{negl}(\lambda)$

$$\Pr\left[\begin{array}{c} b_0 = b_1 = 1 \\ \wedge \\ f_0 \neq f_1 \end{array} \middle| \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda, \mu) \\ (\mathsf{cm}, f_0, t_0, f_1, t_1) \leftarrow \mathcal{A}(\mathsf{gp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f_0, t_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f_1, t_1) \end{array} \right].$$

PCS *is extractable if for any PPT adversary* $\mathcal{A}$*, there exists a probabilistic extractor* $\mathcal{E}$ *running in expected polynomial time such that, given oracle access to* $\mathcal{A}$*, the probability that* $\mathcal{V}$ *accepts* $(\mathsf{cm}, \mathbf{z}, y)$ *but* $\mathcal{E}$ *fails to output* (f, t) *satisfying* $f(\mathbf{z}) = y \wedge \mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f, t) = 1$ *is negligible.*

For other related definitions as SNARK or interactive oracle proof of proximity, please refer to App. A.

3. Veloz: A Distribution Framework for Interleaved-Code-Based Multilinear PCS

In this section, we introduce Veloz, a distribution framework for interleaved-code-based multilinear PCS, and its two instantiations, $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$. As in Sec. 1.2, we first distribute a partial SumCheck reducing μ' variables for μ-variate polynomials in Sec. 3.1. Then, we show a distributed protocol proving vector-matrix multiplication in Sec. 3.2. In Sec. 3.3, we combine these components and present a distribution framework, Veloz. Finally, in Sec. 3.4, we introduce $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$, and in particular design an efficient verification algorithm for $\mathsf{Veloz}_{\mathsf{RS}}$.

System model. In the following, we consider $M = 2^m$ sub-provers $\{\mathcal{P}_i\}_{i=0}^{M-1}$ executing the proof generation procedure, and index each $\mathcal{P}_i$ by $\mathbf{bin}(i) := (b_{m-1}, \cdots, b_0)$, where $i = \sum_{j=0}^{m-1} 2^j b_j$. $\mathcal{P}_0$ acts as the coordinator for aggregation and broadcast, while other sub-provers may communicate directly with one another rather than only through $\mathcal{P}_0$. The number M is fixed and public, as it determines the row-wise partition and the commitment structure for verification. Following the model adopted in prior works (e.g., DIZK [27], deVirgo [12], and HyperPianist [30]), we assume honest sub-provers that correctly follow the prescribed protocol.

3.1. Distributed Partial SumCheck

Given multilinear polynomials $f, z \in \mathcal{F}_\mu^{(\leq 1)}$, a μ'-round partial SumCheck between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ reduces the claim $y = \sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) \cdot z(\mathbf{b})$ to $y' = \sum_{\mathbf{b} \in B_{\mu-\mu'}} f(\mathbf{r}, \mathbf{b}) \cdot z(\mathbf{r}, \mathbf{b})$. For brevity, we show how to distribute a partial SumCheck solely for f. The case $g = f \cdot z$ easily follows, so we later directly present it in Protocol 1.

Recall from Sec. 2 that in the k-th round of SumCheck where $1 \leq k \leq \mu'$, an intermediate claim $s_{k-1}(\mathbf{r}[k-1]) = \sum_{\mathbf{b} \in B_{\mu-k+1}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], \mathbf{b})$ is reduced to the next one by letting $\mathcal{P}$ compute $s_k(x_k) = \sum_{\mathbf{b} \in B_{\mu-k}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b})$ for $\mathcal{V}$, and receive the randomness $\mathbf{r}[k]$. Previous distribution strategies [12], [14], [30], [31] mandate that each sub-prover $\mathcal{P}_i$ compute $s_k^{(i)}(x_k) = \sum_{\mathbf{b} \in B_{\mu-k-m}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b}, \mathbf{bin}(i))$ and delegates $\mathcal{P}_0$ to gather $\{s_k^{(i)}(x_k)\}_{i=0}^{M-1}$, sum them up for $s_k(x_k)$, and send it to $\mathcal{V}$. However, this gathering incurs $O(M)$ additional workload for $\mathcal{P}_0$ each time when SumCheck is invoked, hence not suitable in our case since multiple such invocations are required. Our observation is that the immediate gathering can be *buffered*, reducing $O(M)$ to $O(m)$ in each invocation. To be more specific, instead of letting $\mathcal{P}_0$ directly gather $\{s_k^{(i)}(x_k)\}_{i=1}^{M-1}$, we iteratively require that

1. For $i < \frac{M}{2}$, $\mathcal{P}_i$ acquires $s_k^{(i+\frac{M}{2})}(x_k)$ from $\mathcal{P}_{i+\frac{M}{2}}$, and updates $s_k^{(i)}(x_k) \leftarrow s_k^{(i)}(x_k) + s_k^{(i+\frac{M}{2})}(x_k)$;
2. For $i < \frac{M}{4}$, $\mathcal{P}_i$ acquires $s_k^{(i+\frac{M}{4})}(x_k)$ from $\mathcal{P}_{i+\frac{M}{4}}$, and updates $s_k^{(i)}(x_k) \leftarrow s_k^{(i)}(x_k) + s_k^{(i+\frac{M}{4})}(x_k)$;

$\vdots$

Protocol 1: Distributed partial SumCheck. Given two multilinear polynomials $f, z \in \mathcal{F}_\mu^{(\leq 1)}$, write $g = f \cdot z \in \mathcal{F}_\mu^{(\leq 2)}$. Assuming $\mu' \geq m$ and $\mu \geq \mu' + m$, the claim $y = \sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) \cdot z(\mathbf{b})$ is reduced to $y' = \sum_{\mathbf{b} \in B_{\mu-\mu'}} f(\mathbf{r}, \mathbf{b}) \cdot z(\mathbf{r}, \mathbf{b})$ as follows.

1. For all $\mathbf{b}_1 \in B_{\mu-\mu'-m}$, each $\mathcal{P}_i$ computes $f_{\mathbf{b}_1}^{(i)}(\mathbf{x}) = f(\mathbf{x}, \mathbf{b}_1, \mathbf{bin}(i))$. Similarly for $z_{\mathbf{b}_1}^{(i)}(\mathbf{x})$.
2. In round $1 \leq k \leq \mu'$, reduce evaluation as follows.

- For all $\mathbf{b}_1 \in B_{\mu-\mu'-m}$ and $\mathbf{b}_2 \in B_{\mu'-k}$, each $\mathcal{P}_i$ computes $s_{k,f,\mathbf{b}_1,\mathbf{b}_2}^{(i)}(x_k) = (1-x_k) \cdot f_{\mathbf{b}_1}^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], 0, \mathbf{b}_2) + x_k \cdot f_{\mathbf{b}_1}^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], 1, \mathbf{b}_2)$. Similarly for $s_{k,z,\mathbf{b}_1,\mathbf{b}_2}^{(i)}(x_k)$. Then each $\mathcal{P}_i$ calculates $s_k^{(i)}(x_k) = \sum_{\mathbf{b}_1,\mathbf{b}_2} s_{k,f,\mathbf{b}_1,\mathbf{b}_2}^{(i)}(x_k) \cdot s_{k,z,\mathbf{b}_1,\mathbf{b}_2}^{(i)}(x_k)$.
- For each $j = 1, \cdots, m$, for each $i < 2^{m-j}$, $\mathcal{P}_i$ acquires $s_k^{(i+2^{m-j})}(x_k)$ from $\mathcal{P}_{i+2^{m-j}}$, and updates $s_k^{(i)}(x_k) \leftarrow s_k^{(i)}(x_k) + s_k^{(i+2^{m-j})}(x_k)$. Then, $\mathcal{P}_0$ sends $s_k(x_k) = s_k^{(0)}(x_k)$ to $\mathcal{V}$, and $\mathcal{V}$ verifies $s_k(0) + s_k(1) = s_{k-1}(\mathbf{r}[k-1])$ $\big(s_1(0) + s_1(1) = y$ when $k = 1\big)$. If verification passes, $\mathcal{V}$ responds with $\mathbf{r}[k] \leftarrow_{\$} \mathbb{F}$.
- Each $\mathcal{P}_i$ fixes $x_k = \mathbf{r}[k]$ in $f_{\mathbf{b}_1}^{(i)}$ and $z_{\mathbf{b}_1}^{(i)}$, for all $\mathbf{b}_1 \in B_{\mu-\mu'-m}$.

3. $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ set $y' := s_{\mu'}(\mathbf{r}[\mu'])$.

Figure 1: Distributed partial SumCheck

m. $\mathcal{P}_0$ acquires $s_k^{(1)}(x_k)$ from $\mathcal{P}_1$, updates $s_k^{(0)}(x_k) \leftarrow s_k^{(0)}(x_k) + s_k^{(1)}(x_k)$, and sets $s_k(x_k) = s_k^{(0)}(x_k)$.

Such buffering slightly increases the workload for any other $\mathcal{P}_i$ from 0 to $O(m - \lfloor \log(i+1) \rfloor + 1)$, achieving a more uniform distribution. It only adjusts the destinations where polynomials are sent to, hence requiring *no* extra communication cost. The tradeoff is that the $m = \log M$ aggregation levels are sequential, introducing $\log M - 1$ additional synchronization stages per SumCheck round compared with direct gathering at $\mathcal{P}_0$. We analyze this synchronization overhead in Sec. 4 and discuss its practical latency impact in Sec. 5.2. The detailed distribution for partial SumCheck is given in Protocol 1 in Fig. 1.

3.2. Distributed Vector-Matrix Multiplication

To prove the reduced claim for y' from Protocol 1, $\mathcal{P}$ and $\mathcal{V}$ should run a vector-matrix multiplication protocol for the validity of $\mathbf{u} = \mathbf{r}_\otimes \cdot \mathbf{U}$, where $\mathbf{U} := \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{f})$ and $\mathbf{r}_\otimes := \otimes_{i=1}^{\mu'}(1 - \mathbf{r}[i], \mathbf{r}[i]) \in \mathbb{F}^{2^{\mu'}}$. We now show its distribution, consisting of that for encoding, commitment and querying, and linear combination.

Distributed Encoding. Given a matrix $\mathbf{U} \in \mathbb{F}^{2^{\mu'} \times 2^{\mu-\mu'}}$, to encode $\mathbf{U}$ with a generator matrix $\mathbf{G} \in \mathbb{F}^{2^{\mu-\mu'} \times \frac{2^{\mu-\mu'}}{\rho}}$ of a linear code $\mathcal{C}$, each sub-prover $\mathcal{P}_i$ encodes $2^{\mu'-m}$ rows of $\mathbf{U}$, i.e., computing $\hat{\mathbf{U}}^{(i)} := \mathbf{U}[i \cdot 2^{\mu'-m} : (i+1) \cdot 2^{\mu'-m}, \cdot] \cdot \mathbf{G}$,

Protocol 2: Distributed commitment and querying.

Commitment. Given an encoded matrix $\hat{\mathbf{U}} \in \mathbb{F}^{2^{\mu'} \times \frac{2^{\mu-\mu'}}{\rho}}$ with code rate ρ, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ compute $\mathsf{MC}(\hat{\mathbf{U}})$ as follows.

1. Each $\mathcal{P}_i$ hashes $\hat{\mathbf{U}}^{(i)} = \hat{\mathbf{U}}[i \cdot 2^{\mu'-m} : (i+1) \cdot 2^{\mu'-m}, \cdot]$ column-wise, and obtains a vector $\mathbf{h}^{(i)}$ of $\frac{2^{\mu-\mu'}}{\rho}$ values. Then, $\mathcal{P}_i$ sends $\mathbf{h}^{(i)}[j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$ to $\mathcal{P}_j$ for all $j \neq i$.
2. For each $j \in [0 : M]$, $\mathcal{P}_j$ computes $\mathbf{h}[k] = \mathbb{H}(\mathbf{h}^{(0)}[k], \cdots, \mathbf{h}^{(M-1)}[k])$ for all $k \in [j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$, and $\mathsf{MC}(\mathbf{h}[j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}])$.
3. $\mathcal{P}_0$ collects the Merkle roots, and computes $\mathsf{MC}(\hat{\mathbf{U}})$.

Querying. On receiving $\mathcal{V}$'s indexed set $I \subseteq [0 : \frac{2^{\mu-\mu'}}{\rho}]$,

1. $\mathcal{P}_0$ collects the columns $(\hat{\mathbf{U}}^{(i)})_I$ from each $\mathcal{P}_i$, concatenates them to obtain $(\hat{\mathbf{U}})_I$, and sends $(\hat{\mathbf{U}})_I$ and $|I|$ Merkle paths to $\mathcal{V}$.
2. For each index $p \in I$, $\mathcal{V}$ computes $\mathbf{h}[p]$ with the column $\hat{\mathbf{U}}[\cdot, p]$, and verifies its Merkle path to $\mathsf{MC}(\hat{\mathbf{U}})$.

Figure 2: Distributed commitment and querying

without communication. The simplicity of this process essentially originates from our interleaved code-specific proof aggregation method, which separates the relations among codewords. Looking ahead, it greatly saves sub-prover communication to align preimages of Merkle leaves.

Distributed commitment and querying. For a single prover, to prove row-wise that $\hat{\mathbf{U}} = \mathbf{U} \cdot \mathbf{G} \in \mathbb{F}^{2^{\mu'} \times \frac{2^{\mu-\mu'}}{\rho}}$ is close to the code space $\mathcal{C}$, $\mathcal{P}$ computes and sends $\mathcal{V}$ a Merkle commitment $\mathsf{MC}(\hat{\mathbf{U}})$. Then $\mathcal{V}$ queries for $\Theta(\lambda)$ randomly chosen columns of $\hat{\mathbf{U}}$, and checks their corresponding Merkle paths. To commit to $\hat{\mathbf{U}}$, a standard technique from [8] proceeds by first hashing $\hat{\mathbf{U}}$ column-wise, and then building a Merkle tree for the derived row of hash values. When $\mathcal{V}$ queries for columns, they are sent in full as the preimages of Merkle leaves, so proof size is $O(\lambda \cdot (2^{\mu'} + \log \frac{2^{\mu-\mu'}}{\rho}))$.

Nevertheless, this technique does not directly apply to the distributed setting: after $\mathcal{P}_i$ hashes $\hat{\mathbf{U}}^{(i)} = \hat{\mathbf{U}}[i \cdot 2^{\mu'-m} : (i+1) \cdot 2^{\mu'-m}, \cdot]$ column-wise and obtains a vector $\mathbf{h}^{(i)}$ of $\frac{2^{\mu-\mu'}}{\rho}$ hash values, it gets stuck. The reason is the information of each column of $\hat{\mathbf{U}}$ is evenly scattered among all sub-provers. To keep the commitment on track, one can let each $\mathcal{P}_i$ directly build a Merkle tree for $\mathbf{h}^{(i)}$, and delegate $\mathcal{P}_0$ to gather and send $\{\mathsf{MC}(\mathbf{h}^{(i)})\}_{i=0}^{M-1}$ to $\mathcal{V}$. The resulting proof size amounts to $O(\lambda \cdot (2^{\mu'} + M \cdot \log \frac{2^{\mu-\mu'}}{\rho}))$, whose principal part, $2^{\mu'}$, is identical to that generated by a single prover. However, the term $M \cdot \log \frac{2^{\mu-\mu'}}{\rho}$ usually outgrows $2^{\mu'}$ in large-scale computation. For example, given $\mu = 30$, $\mu' = 5$, $\rho = \frac{1}{4}$ and $M = 16$, the proof size already grows by a factor of 8, let alone the cases $M \geq 32$ in practical use. Therefore, removing the $O(M)$ overhead is a necessity.

Protocol 3: Distributed linear combination. Given a matrix $\mathbf{U} \in \mathbb{F}^{2^{\mu'} \times 2^{\mu-\mu'}}$ and a vector $\mathbf{r} \in \mathbb{F}^{\mu'}$, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ compute $\mathbf{u} = \mathbf{r}_{\otimes} \cdot \mathbf{U}$ as follows.

1. Each $\mathcal{P}_i$ computes $\mathbf{r}_{\otimes}^{(i)} = \tilde{eq}_{\mathbf{bin}(i)}(r[1], \cdots, r[m]) \cdot \otimes_{j=m+1}^{\mu'}(1 - r[j], r[j]) \in \mathbb{F}^{2^{\mu'-m}}$ based on its index i, and $\mathbf{u}^{(i)} = \mathbf{r}_{\otimes}^{(i)} \cdot \mathbf{U}^{(i)} \in \mathbb{F}^{2^{\mu-\mu'}}$.
2. For each $j \neq i$, $\mathcal{P}_i$ sends $\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$ to $\mathcal{P}_j$. Then, each $\mathcal{P}_j$ adds the M vectors and obtains $\mathbf{u}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$.

Figure 3: Distributed linear combination

We observe that M exists because columns are not completely hashed: $\mathbf{h}^{(i)}$ is directly used as leaves. To address this, we force a complete hashing as follows. Let each $\mathcal{P}_i$ send $\mathcal{P}_j$ $\mathbf{h}^{(i)}[j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$; $\mathcal{P}_j$ then hashes M elements $\mathbf{h}[k] := \mathbb{H}(\mathbf{h}^{(0)}[k], \cdots, \mathbf{h}^{(M-1)}[k])$, for all $k \in [j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$. Now, with $\mathbf{h} = (\mathbf{h}[0], \cdots, \mathbf{h}[\frac{2^{\mu-\mu'}}{\rho} - 1])$ evenly scattered among $\{\mathcal{P}_i\}_{i=0}^{M-1}$, each $\mathcal{P}_j$ computes $\mathsf{MC}(\mathbf{h}[j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}])$. Finally, $\mathcal{P}_0$ gathers these M roots, treats them as leaves, and computes $\mathsf{MC}(\hat{\mathbf{U}})$. The proof size now reduces to $O(\lambda \cdot (2^{\mu'} + \log \frac{2^{\mu-\mu'}}{\rho}))$, with extra $O(M \cdot \frac{2^{\mu-\mu'}}{\rho})$ communication. For details, please refer to Protocol 2 in Fig. 2.

Distributed Linear Combination. After commitment, the prover $\mathcal{P}$ linearly combines the rows of $\mathbf{U}$ with logarithmic randomness $\mathbf{r} = (r[1], \cdots, r[\mu'])$ from partial SumCheck, i.e., computing $\mathbf{u} = \mathbf{r}_{\otimes} \cdot \mathbf{U}$. For distribution, let each $\mathcal{P}_i$ compute $\mathbf{u}^{(i)} = \mathbf{r}_{\otimes}^{(i)} \cdot \mathbf{U}^{(i)} \in \mathbb{F}^{2^{\mu-\mu'}}$. It then suffices for $\mathcal{P}_0$ to gather $\{\mathbf{u}^{(i)}\}_{i=0}^{M-1}$ for summation. However, the derived $\mathbf{u}$ will be reshaped into $\mathbf{U}' = \mathbf{Mat}_{2^{\mu''} \times 2^{\mu-\mu'-\mu''}}(\mathbf{u})$ for the next iteration, requiring $\mathcal{P}_0$ to broadcast $\mathbf{u}$ to other sub-provers. The total communication therefore doubles to $2M \cdot 2^{\mu-\mu'}$. Though this is optimal in complexity since each $\mathcal{P}_i$ has to join communication, the constant overhead is redundant because of the gather-and-broadcast pattern, leaving room for further communication savings.

Our observation is that, during the row-wise linear combination, each column of $\mathbf{U}$ is "squeezed" into a field element. This scenario is quite similar to that in the distributed commitment phase above, where each column of $\hat{\mathbf{U}}$ is hashed into one value. Thus, we can mimic the aforementioned element exchanging process as follows. Let each $\mathcal{P}_i$ send $\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$ to $\mathcal{P}_j$. Then, each $\mathcal{P}_j$ sums $\{\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]\}_{i=0}^{M-1}$ and obtains $\mathbf{u}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$. After rearrangement, communication halves to $M \cdot 2^{\mu-\mu'}$. Coming with this reduction is another advantage: we obtain $\mathbf{U}'$ *for free*, as $\mathbf{u}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$ is exactly the row-major concatenation of $\mathbf{U}'[j \cdot 2^{\mu''-m} : (j+1) \cdot 2^{\mu''-m}, \cdot]$ that will be used in the next iteration. For details of distributed linear combination, please refer to Protocol 3 in Fig. 3.

3.3. Putting Everything Together

Now, we combine all the building blocks to present our distribution framework, Veloz, for multilinear PCS. In each round of iteration below, denote by Enc_k the encoding algorithm instantiated with an efficient linear code $\mathcal{C}_k$, including RS code and Brakedown code [8].

$\mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda, \mu)$: on input the security parameter λ and the number of variables $\mu \in \mathbb{N}$, it outputs public parameters $\mathsf{gp} = (M, \ell, \{\mu'_k, \mu_k, \mathbf{G}_k\}_{k=1}^{\ell-1})$. Here M is the sub-prover number, $\ell \geq 2$ the iteration number, and $\{\mu'_k, \mu_k\}_{k=1}^{\ell-1}$ are a sequence of dimensions satisfying $\mu'_{k+1} + \mu_{k+1} = \mu_k$ for $k \in [1 : \ell - 1]$ (we write $\mu_0 := \mu = \mu'_1 + \mu_1$). Each $\mathbf{G}_k \in \mathbb{F}^{2^{\mu_k} \times \frac{2^{\mu_k}}{\rho}}$ is the generator matrix of a $[\frac{2^{\mu_k}}{\rho}, 2^{\mu_k}, d_k]$-code $\mathcal{C}_k$ with relative minimum distance $\delta_{\mathcal{C}_k} = \frac{\rho \cdot d_k}{2^{\mu_k}}$.

$(\mathsf{cm}; t) \leftarrow \mathsf{Commit}(\mathsf{gp}, f)$: on input public parameters gp and a multilinear polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$ with Lagrange coefficients $\mathbf{f} = f(B_{\mu_0}) = \mathbf{u}_0 \in \mathbb{F}^{2^{\mu_0}}$, it reshapes $\mathbf{U}_1 = \mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{u}_0)$, encodes $\hat{\mathbf{U}}_1 = \mathsf{Enc}_1(\mathbf{U}_1)$, and outputs commitment $\mathsf{cm} = \mathsf{MC}(\hat{\mathbf{U}}_1)$ and a *full* opening hint $t = \hat{\mathbf{U}}_1$.

$b \leftarrow \mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f, t)$: on input the public parameters gp, a commitment cm, a multilinear polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$, and a *partial* opening hint t a collection of distinct Merkle preimages against cm, missing some columns indexed by $\mathcal{M} \subset [0 : \frac{2^{\mu_1}}{\rho}]$ [2], it reshapes the Lagrange coefficients $\mathbf{f} \in \mathbb{F}^{2^{\mu_0}}$ as $\mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{f})$, verifies all Merkle paths against cm, and outputs a bit $b = 1$ if $|\Delta^{\mu'_1}_{\mathcal{C}_1}(\mathsf{Enc}_1(\mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{f})), t) \cup \mathcal{M}| < \frac{d_1}{2}$ (recall that $\Delta^{\mu'_1}_{\mathcal{C}_1}(\cdot, \cdot)$ denotes the set of column indices that two matrices differ within the μ'_1-fold interleaved code $\mathcal{C}_1^{\mu'_1}$); otherwise it outputs $b = 0$.

$b \leftarrow \mathsf{Eval}(\mathsf{gp}, \mathsf{cm}, \mathbf{z}, y_1; f, t)$: this is a distributed evaluation protocol between sub-provers $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and a verifier $\mathcal{V}$. On input public parameters gp, a commitment cm, an evaluation point $\mathbf{z} \in \mathbb{F}^\mu$ and a value $y_1 \in \mathbb{F}$, with $\{\mathcal{P}_i\}_{i=0}^{M-1}$ additionally knowing a multilinear polynomial f and its opening hint t, $\mathcal{V}$ outputs a bit b at the end of protocol, denoting acceptance or rejection for the claims that $f(\mathbf{z}) = y_1$ and $\mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f, t) = 1$.

In the evaluation protocol, the claim $y_1 = f(\mathbf{z})$ is first transformed to $y_1 = \langle \mathbf{u}_0, \mathbf{z}_1 \rangle$, where $\mathbf{z}_1 = \mathbf{z}_{\otimes} = \otimes_{i=1}^{\mu_0}(1 - \mathbf{z}[i], \mathbf{z}[i])$. Then in each round, the claim $y_k = \langle \mathbf{u}_{k-1}, \mathbf{z}_k \rangle$ is iteratively reduced to $y_{k+1} = \langle \mathbf{u}_k, \mathbf{z}_{k+1} \rangle$, which interleaves the distributed partial SumCheck in Sec. 3.1 and the distributed vector-matrix multiplication in Sec. 3.2. We formally present the protocol in Fig. 4, where the iteratively reduced claims are highlighted in blue.

2. Following [38, Construction 3], Commit outputs an encoded matrix, while Open may receive a partial opening hint, with a set $\mathcal{M}$ of columns left unspecified, and view the complete matrix as the case $\mathcal{M} = \emptyset$. We prove extractability under this setting in Sec. 4 and App. B. As in [38], the extractor recovers a polynomial and an opening hint, instead of complete Merkle preimages. This extracted witness can be leveraged by outer compositions whose security reductions access the PCS through its opening and evaluation relations and do not require recovery of Merkle preimages for the unspecified columns.

Protocol 4: Distributed evaluation protocol $\mathsf{Eval}(\mathsf{gp}, \mathsf{cm}, \mathbf{z}, y_1; f, t)$.
Public input: a commitment $\mathsf{cm} = \mathsf{MC}(\hat{\mathbf{U}}_1)$, an evaluation point $\mathbf{z} \in \mathbb{F}^{\mu_0}$, and a purported value $y_1 \in \mathbb{F}$.
Prover witness: a multilinear polynomial $f \in \mathcal{F}_{\mu_0}^{(\leq 1)}$ and $t = \hat{\mathbf{U}}_1 = \mathsf{Enc}_1(\mathbf{U}_1)$.
With $\mathbf{u}_0 = \mathbf{f}$, $\mathbf{U}_1 = \mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{u}_0)$, and $\mathbf{z}_1 = \mathbf{z}_\otimes$, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ convince $\mathcal{V}$ that $y_1 = f(\mathbf{z}) = \langle \mathbf{u}_0, \mathbf{z}_1 \rangle$ as follows.

1. In round $1 \leq k \leq \ell - 2$, the claim $y_k = \langle \mathbf{u}_{k-1}, \mathbf{z}_k \rangle$ is reduced to $y_{k+1} = \langle \mathbf{u}_k, \mathbf{z}_{k+1} \rangle$, with $\{\mathcal{P}_i\}_{i=0}^{M-1}$ knowing $\mathbf{U}_k = \mathbf{Mat}_{2^{\mu' k} \times 2^{\mu_k}}(\mathbf{u}_{k-1})$ and $\mathcal{V}$ knowing $\mathsf{MC}(\hat{\mathbf{U}}_k)$ (for $k = 1$, $\mathsf{MC}(\hat{\mathbf{U}}_1) = \mathsf{cm}$):
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ run a μ'_k-round distributed partial SumCheck for $f(\mathbf{r}_1 \| \cdots \| \mathbf{r}_{k-1}, \cdot)$ and $z_k(\cdot) \in \mathcal{F}_{\mu_{k-1}}^{(\leq 1)}$ as in Protocol 1, where z_k is the multilinear polynomial interpolated by $\mathbf{z}_k$. After that, both sides obtain randomness $\mathbf{r}_k \in \mathbb{F}^{\mu'_k}$ and the reduced value $y'_k \in \mathbb{F}$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ invoke Protocol 3 to compute $\mathbf{u}_k = (\mathbf{r}_k)_\otimes \cdot \mathbf{U}_k \in \mathbb{F}^{2^{\mu_k}}$, and reshape it into $\mathbf{U}_{k+1} = \mathbf{Mat}_{2^{\mu'_{k+1}} \times 2^{\mu_{k+1}}}(\mathbf{u}_k)$. Then, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ encode $\mathbf{U}_{k+1}$ to get $\hat{\mathbf{U}}_{k+1} = \mathsf{Enc}_{k+1}(\mathbf{U}_{k+1})$, and invoke the commitment phase from Protocol 2 to obtain and send $\mathcal{V}$ the commitment $\mathsf{MC}(\hat{\mathbf{U}}_{k+1})$, which will be queried in round $k+1$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ invoke the querying phase from Protocol 2, where $\mathcal{V}$ samples $I_k \subseteq [0 : \frac{2^{\mu_k}}{\rho}]$ and queries for $(\hat{\mathbf{U}}_k)_{I_k}$, the $|I_k|$ indexed columns of $\hat{\mathbf{U}}_k$. Then $\mathcal{V}$ verifies the Merkle paths of $(\hat{\mathbf{U}}_k)_{I_k}$. If verification passes, $\mathcal{V}$ samples and sends $\gamma_k \leftarrow_\$ \mathbb{F}$ to $\{\mathcal{P}_i\}_{i=0}^{M-1}$, and both sides keep $\mathbf{s}_k := (1, \gamma_k, \cdots, \gamma_k^{|I_k|}) \in \mathbb{F}^{|I_k|+1}$.
- Both sides compute $\mathbf{w}_k = (\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_{2^{\mu'_k} \times 2^{\mu_k}}(\mathbf{z}_k) \in \mathbb{F}^{2^{\mu_k}}$, and $\mathbf{z}_{k+1} = ((\mathbf{G}_k)_{I_k} \| \mathbf{w}_k^\top) \cdot \mathbf{s}_k \in \mathbb{F}^{2^{\mu_k}}$. $\mathcal{V}$ additionally computes $\mathbf{v}_k = (\mathbf{r}_k)_\otimes \cdot (\hat{\mathbf{U}}_k)_{I_k} \in \mathbb{F}^{|I_k|}$, and $y_{k+1} = \langle \mathbf{v}_k \| y'_k, \mathbf{s}_k \rangle$. The claim now is reduced to $y_{k+1} = \langle \mathbf{u}_k, \mathbf{z}_{k+1} \rangle$.

2. In round $k = \ell - 1$, the claim is $y_{\ell-1} = \langle \mathbf{u}_{\ell-2}, \mathbf{z}_{\ell-1} \rangle$, with $\{\mathcal{P}_i\}_{i=0}^{M-1}$ knowing $\mathbf{U}_{\ell-1}$, and $\mathcal{V}$ knowing $\mathsf{MC}(\hat{\mathbf{U}}_{\ell-1})$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ run a $\mu'_{\ell-1}$-round distributed partial SumCheck for $f(\mathbf{r}_1 \| \cdots \| \mathbf{r}_{\ell-2}, \cdot)$ and $z_{\ell-1}(\cdot) \in \mathcal{F}_{\mu_{\ell-2}}^{(\leq 1)}$ as in Protocol 1. After that, both sides obtain randomness $\mathbf{r}_{\ell-1} \in \mathbb{F}^{\mu'_{\ell-1}}$ and the reduced value $y'_{\ell-1} \in \mathbb{F}$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ invoke Protocol 3 to compute $\mathbf{u}_{\ell-1} = (\mathbf{r}_{\ell-1})_\otimes \cdot \mathbf{U}_{\ell-1} \in \mathbb{F}^{2^{\mu_{\ell-1}}}$, and directly send it to $\mathcal{V}$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ invoke the querying phase from Protocol 2, where $\mathcal{V}$ samples $I_{\ell-1} \subseteq [0 : \frac{2^{\mu_{\ell-1}}}{\rho}]$ and queries for $(\hat{\mathbf{U}}_{\ell-1})_{I_{\ell-1}}$. Then $\mathcal{V}$ verifies the Merkle paths of $(\hat{\mathbf{U}}_{\ell-1})_{I_{\ell-1}}$. If verification passes, $\mathcal{V}$ computes $\mathbf{w}_{\ell-1} = (\mathbf{r}_{\ell-1})_\otimes \cdot \mathbf{Mat}_{2^{\mu'_{\ell-1}} \times 2^{\mu_{\ell-1}}}(\mathbf{z}_{\ell-1})$. The claim now is reduced to (1) $\mathbf{u}_{\ell-1} \cdot (\mathbf{G}_{\ell-1})_{I_{\ell-1}} = (\mathbf{r}_{\ell-1})_\otimes \cdot (\hat{\mathbf{U}}_{\ell-1})_{I_{\ell-1}}$ $\wedge$ (2) $y'_{\ell-1} = \langle \mathbf{w}_{\ell-1}, \mathbf{u}_{\ell-1} \rangle$, for which $\mathcal{V}$ directly verifies and outputs $b = 1$ or 0.

Figure 4: Distributed evaluation protocol

3.4. Instantiations with RS and Brakedown codes

We now instantiate Veloz with RS code and the fast linear code from Brakedown [8], and obtain $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$, respectively. The metrics including proving time, communication and proof size can be analyzed independent from code choices, so we summarize them in Sec. 4. What we should focus on, however, is the different verification efficiency of $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$, as we now explain.

Note that the current verification in Protocol 4 is not succinct regardless of distribution: computing $\mathbf{w}_1 = (\mathbf{r}_1)_\otimes \cdot \mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{z}_1)$ requires $O(2^{\mu'_1 + \mu_1}) = O(|f|)$ time. As pointed out in [34], a possible solution is to exploit the following structure: given $\mathbf{z}_1 = \mathbf{z}_\otimes$, $\mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{z}_1) =$

$$\otimes_{i=1}^{\mu'_1}(1 - \mathbf{z}[i], \mathbf{z}[i])^\top \cdot \otimes_{i=\mu'_1+1}^{\mu'_1+\mu_1}(1 - \mathbf{z}[i], \mathbf{z}[i]). \quad (1)$$

Recall from Sec. 2.1 that, given two vectors with identical tensor structure, their inner product can be computed efficiently. Thus, $(\mathbf{r}_1)_\otimes \cdot \mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{z}_1) =$

$$\prod_{i=1}^{\mu'_1} \langle (1 - \mathbf{r}_1[i], \mathbf{r}_1[i]), (1 - \mathbf{z}[i], \mathbf{z}[i]) \rangle \bigotimes_{i=\mu'_1+1}^{\mu'_1+\mu_1} (1 - \mathbf{z}[i], \mathbf{z}[i]). \quad (2)$$

The verification cost for computing $\mathbf{w}_1$ now reduces to $O(\mu'_1 + 2^{\mu_1}) = o(|f|)$, so both $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$ are succinct.

That said, when $k \geq 2$, the tensor structure of $\mathbf{z}_k$ is spoiled by random linear combination of columns of $(\mathbf{G}_{k-1})_{I_{k-1}}$ and $\mathbf{w}_{k-1}^\top$, so the previous technique only applies to the first round in $\mathsf{Veloz}_{\mathsf{Lin}}$. However, we note that the same technique still applies to $\mathsf{Veloz}_{\mathsf{RS}}$ in each round of iteration to further improve verification efficiency, as unfolded below. Since $\mathbf{z}_k = ((\mathbf{G}_{k-1})_{I_{k-1}} \| \mathbf{w}_{k-1}^\top) \cdot \mathbf{s}_{k-1} \in \mathbb{F}^{2^{\mu_{k-1}}}$ generates $\mathbf{w}_k = (\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_{2^{\mu'_k} \times 2^{\mu_k}}(\mathbf{z}_k) \in \mathbb{F}^{2^{\mu_k}}$, our core observation is the two operators, $\mathbf{Mat}(\cdot)$ and $\cdot \mathbf{s}_k$, are *compatible*. This means $\mathcal{V}$ can compute $(\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}(\mathbf{g})$ for each column $\mathbf{g}$ of $(\mathbf{G}_{k-1})_{I_{k-1}} \| \mathbf{w}_{k-1}^\top$, before linearly combining them with $\mathbf{s}_k$. By equation 2, verification would be more efficient if $\mathbf{g}$ has a tensor structure as $(\mathbf{r}_k)_\otimes$ does. Fortunately, this condition holds. It is well-known that a RS code generator matrix $\mathbf{G} \in \mathbb{F}^{s \times t}$ can be linearly transformed to a *Vandermonde matrix*, i.e., each column $\mathbf{G}[\cdot, i]$ is of the form $(1, g_i, \cdots, g_i^{s-1})^\top$. Assuming s is a power of two, $\mathbf{G}[\cdot, i] = ((1, g_i^{\frac{s}{2}}) \otimes (1, g_i^{\frac{s}{4}}) \otimes \cdots \otimes (1, g_i))^\top$. Therefore, we can transform $\mathbf{g}$ by equation 1, and obtain $(\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}(\mathbf{g})$ using equation 2, further simplifying verification. For detailed design, please refer to Fig. 5.

Efficient verification. In the following, we use $\mathbf{Mat}_k$ instead of $\mathbf{Mat}_{2^{\mu'_k} \times 2^{\mu_k}}$ for brevity. In round k, for each $i \in I_k$, denote the i-th column of $(\mathbf{G}_k)_{I_k}$ by $\mathbf{g}_{k,i} = (1, g_{k,i}, \cdots, g_{k,i}^{2^{\mu_k}-1})^\top$.

1. In round $k = 1$, computes $\mathbf{w}_1 = (\mathbf{r}_1)_\otimes \cdot \mathbf{Mat}_1(\mathbf{z}_1) = \prod_{i=1}^{\mu'_1} \langle (1-r_i, r_i), (1-z_i, z_i) \rangle \cdot \otimes_{i=\mu'_1+1}^{\mu'_1+\mu_1} (1-z_i, z_i)$.
2. In round $k \geq 2$, computes

$$\begin{aligned}
\mathbf{w}_k &= (\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_k(\mathbf{z}_k) = (\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_k\big(((\mathbf{G}_{k-1})_{I_{k-1}} \| \mathbf{w}_{k-1}^\top) \cdot \mathbf{s}_{k-1}\big) \\
&= (\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_k\big((\mathbf{g}_{k-1,1} \| \cdots \| \mathbf{g}_{k-1,|I_{k-1}|} \| \mathbf{w}_{k-1}^\top) \cdot \mathbf{s}_{k-1}\big) \\
&= \Big[\big((\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_k(\mathbf{g}_{k-1,1})\big)^\top \| \cdots \| \big((\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_k(\mathbf{g}_{k-1,|I_{k-1}|})\big)^\top \| \big((\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_2(\mathbf{w}_{k-1}^\top)\big)^\top \Big] \cdot \mathbf{s}_{k-1},
\end{aligned}$$

where in the i-th slot, $(\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_k(\mathbf{g}_{k-1,i})^\top = \prod_{j=1}^{\mu'_k} \langle (1-r_j, r_j), (1, g_{k-1,i}^{2^{\mu_{k-1}-j}}) \rangle \cdot \otimes_{j=\mu'_k+1}^{\mu'_k+\mu_k} (1, g_{k-1,i}^{2^{\mu_{k-1}-j}})$.

Figure 5: Efficient verification

Now we analyze verification cost for computing $\mathbf{w}_k$. In round $k = 1$, this requires $O(\mu'_1)$ multiplications over $\mathbb{F}$. In round $k \geq 2$, it requires $O(|I_{k-1}| \cdot \mu'_k)$ (for columns of $(\mathbf{G}_{k-1})_{I_{k-1}}$) + $O((\sum_{j=1}^{k-2} |I_j|) \cdot \mu'_k)$ (for $\mathbf{w}_{k-1}^\top$) field operations. Thus, writing $|I_k| = \Theta(\lambda)$ for all k, verification in Fig. 5 costs $O(\mu'_1 + \lambda \cdot \sum_{k=1}^{\ell-1}(k-1) \cdot \mu'_k)$ field operations.

4. Security and Complexity

Security. As in the system model in Sec 3, we assume that the sub-provers trust each other, and analyze security as in the single-prover setting. In addition, to make our design non-interactive, we apply the Fiat-Shamir heuristic [44] in the random oracle model, to remove the interaction in the evaluation protocol. We have the following theorem.

Theorem 1. *The PCS construction in Sec. 3.3 is complete, binding, and extractable.*

We sketch our proof here, and present a formal one in App. B. For completeness, we reduce it to that in Ligerito [34], where an honest prover passes verification. To argue binding, we assume that it does not hold, and lead to a contradiction based on collision resistance of hash function.

For soundness, we follow the analysis in [34, Sec. 6.3], and bound the soundness error arising from each SumCheck round (Protocol 1), $\mathcal{V}$'s queries to matrix columns (Protocol 2), and batching columns with a random vector (Protocol 4). As to extractability, we prove it in the random oracle model, following the proof of [38, Theorem 5]. We first recursively construct extractors $\{\mathcal{E}_k\}_{k=1}^{\ell-1}$ starting from the final iteration. Then, let each $\mathcal{E}_k$ rewind the adversary and reprogram the random oracle to obtain fresh random evaluation points, and use the witness information extracted from $\mathcal{E}_{k+1}$ to produce an opening witness for cm_k. Applying this from $k = \ell - 1$ to $k = 1$ extracts a desired witness.

Complexity. In the following, we summarize the asymptotic costs of Veloz, and defer detailed calculations to App. C.

(1) **Prover.** The proving cost of each $\mathcal{P}_i$ is dominated by encoding, which in $\mathsf{Veloz}_{\mathsf{RS}}$ is $O(\frac{N}{M} \log \frac{N}{\log N})$, due to FFT-based encoding algorithm and iteration, and in $\mathsf{Veloz}_{\mathsf{Lin}}$ is $O(\frac{N}{M})$ because of the linear time encoding in Brakedown.

(2) **Synchronization.** The number of synchronizations among $\{\mathcal{P}_i\}_{i=0}^{M-1}$ is dominated by the buffered SumCheck, which requires $O(\log M - 1)$ additional sequential synchronization stages per SumCheck round, compared to prior distribution method. Such cost amounts to $O(\log M \cdot \log N)$.

(3) **Communication.** Communication among sub-provers consists of those received SumCheck challenges, the queried columns sent for verification, and column-wise aggregation. In $\mathsf{Veloz}_{\mathsf{RS}}$, they amount to $O(\lambda \cdot \frac{\log^2 N}{\log \log N} + M \cdot \frac{N}{\log N})$, by letting the number of rows in each round be $2^{\mu'_k} = O(\log N)$[3]. In $\mathsf{Veloz}_{\mathsf{Lin}}$, communication is $O(\lambda \cdot K + M \cdot \frac{N}{K})$, where $K \in [\sqrt[3]{N}, \sqrt{N}]$ is the row number of the initial matrix $\mathbf{U}_1$.

(4) **Verifier.** Following Sec. 3.4, the tensor structure in $\mathsf{Veloz}_{\mathsf{RS}}$ produces $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ verification cost, while $\mathsf{Veloz}_{\mathsf{Lin}}$ requires $O(\lambda \cdot (K + \frac{N}{K}))$, due to the need to linearly combine the queried columns without tensor structure.

(5) **Proof size.** The proof size is determined by the columns queried by $\mathcal{V}$ and the vector $\mathbf{u}_{\ell-1}$ sent in the final iteration. In $\mathsf{Veloz}_{\mathsf{RS}}$, it accumulates in each iteration, and achieves $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ in total; while in $\mathsf{Veloz}_{\mathsf{Lin}}$, it is dominated by the parameter in the first round, which yields $O(\lambda \cdot K)$.

5. Implementation and Evaluation

We implement $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$ using 6200+ lines of Rust based on the `arkworks` ecosystem [45]. Source code is available at https://zenodo.org/records/22782023. In Sec. 5.1, we evaluate both schemes on polynomials of sizes $N = 2^{20}$, 2^{22}, 2^{24}, 2^{26}, 2^{28} and 2^{30}. In Sec. 5.2, we compare them to prior code-based distributed PCSs. Since deVirgo [12] has not provided publicly available implementations, we benchmark $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$ against FRIttata [13], HyperFond [14], and $\mathrm{DePIP}_{\mathrm{FRI}}$ [15]. The metrics we consider are proving time, total communication, memory cost of $\mathcal{P}_0$, proof size, and verification time.

Experimental setup. Our experiments are conducted on a Ubuntu 24.04 ecs.g9ae.16xlarge cloud server with 64 vCPUs

3. To ensure that the following proof size is independent of M, it suffices to bound M by the row number $2^{\mu'_k}$. Here, the setting $2^{\mu'_k} = \log N$, which implies $M \leq O(\log N)$, aims to minimize proof size.

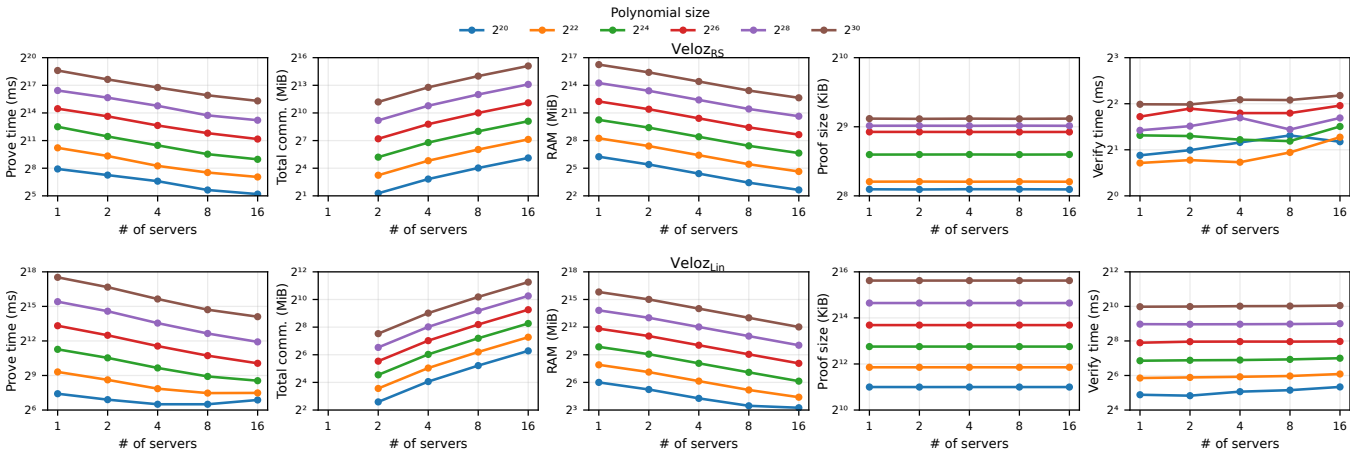

Figure 6: Scalability of Veloz$_{\text{RS}}$ and Veloz$_{\text{Lin}}$

and 256 GiB RAM. For sub-provers, each $\mathcal{P}_i$ is pinned to a virtual CPU core, and we set $M \in \{1, 2, 4, 8, 16\}$, where $M = 1$ is Ligerito [34]. Communication among sub-provers in the distributed setting is simulated in a local area network, with a transfer rate of 10Gbps.

Choices of parameters. Our base field $\mathbb{F}$ is the quadratic extension of the *Goldilocks field* $\mathbb{F}_p$, where $p = 2^{64} - 2^{32} + 1$. Here we adopt an extension from Ligerito [34, Sec. 6.5]: in the first iteration, sub-provers commit to $\hat{\mathbf{U}}_1$ in $\mathbb{F}_p \subseteq \mathbb{F}_{p^2}$, which concretely accelerates proof generation and shrinks proof size. The hash function we incorporate is BLAKE3.

The code rate ρ is set to $\frac{1}{4}$ for Veloz$_{\text{RS}}$ and $\frac{3}{5}$ for Veloz$_{\text{Lin}}$. We set $\lambda = 100$. To guarantee soundness error $\epsilon \leq 2^{-100}$ as per the analysis in Sec. 4, we set $|I_k| = 148$ in Veloz$_{\text{RS}}$ and $|I_k| = 4125$ in Veloz$_{\text{Lin}}$. During evaluation for each $N \in \{2^{20}, \cdots, 2^{30}\}$, we choose $\mu'_k = O(\log \log N) = 5$ or 6 in Veloz$_{\text{RS}}$ to ensure that $2^{\mu_k} \geq 148$ for each $k < \ell - 1$, and $\mu'_k = \frac{\mu_{k-1}}{2}$ in Veloz$_{\text{Lin}}$.

5.1. Scalability Analysis

In this section, we evaluate how Veloz$_{\text{RS}}$ and Veloz$_{\text{Lin}}$ scale with the number of sub-provers M and the polynomial size N. As in Fig. 6, for a fixed N, proving time decreases substantially with M. In particular, at $N = 2^{30}$, Veloz$_{\text{RS}}$ decreases from 399s with $M = 1$ to 61.8s and 40.6s with $M = 8$ and 16, giving $6.46\times$ and $9.83\times$ speedups, respectively. Veloz$_{\text{Lin}}$ similarly decreases from 189s to 26.9s and 17.6s, with respective $7.03\times$ and $10.7\times$ speedups. Note that we simulate sub-provers on a single cloud server, so the marginal acceleration fades as M doubles, due to cache misses and limited memory bandwidth.

For other metrics, increasing M trades higher communication for lower memory usage at $\mathcal{P}_0$, as expected in a distribution scheme. Notably, communication grows less than proportionally with M over the evaluated range, which is consistent with its decomposition into an M-independent component and an M-dependent term, as analyzed in Sec. 4. More importantly, proof size remains stable, and verification exhibits no systematic growth as M increases, empirically confirming that distribution does not introduce an $O(M)$ overhead into proof size or verification. We note here that verification time displays some fluctuations, particularly for Veloz$_{\text{RS}}$, as verification takes only a few milliseconds, so small variations due to system noise, CPU frequency, and cache state might become relatively visible at this scale.

The growth with N also agrees with the complexity analysis. Concretely, fixing $M = 8$, increasing N from 2^{28} to 2^{30} raises Veloz$_{\text{Lin}}$'s proving time from 6.42s to 26.9s ($4.19\times$), closely matching the $O(\frac{N}{M})$ complexity. In the same interval, Veloz$_{\text{RS}}$ increases from 13.6s to 61.8s ($4.5\times$), consistent with its additional logarithmic overhead. Finally, the verification time of Veloz$_{\text{RS}}$ is not strictly monotonic as N increases, which is caused by the concrete choices of μ_k across different N. In our setting, we ensure $\mu_{\ell-2} \geq \lceil \log 148 \rceil > \mu_{\ell-1}$, so $\mathbf{u}_{\ell-1}$ can be sent directly to $\mathcal{V}$ in the final round. Particularly, the μ_k's for $N = 2^{20}$ are 20, 14, 8, and those for $N = 2^{22}$ are 22, 16, 11, 6, so $\mathbf{u}_{\ell-1}$ is shorter in the latter case, hence faster verification. These parameters can be flexibly adjusted to suit practical requirements.

5.2. Comparing to Prior Works

In this subsection, we compare Veloz$_{\text{RS}}$ and Veloz$_{\text{Lin}}$ to FRIttata [13], HyperFond [14], and DePIP$_{\text{FRI}}$ [15] on the same device, with $M = 8$, $\rho = \frac{1}{4}$, and $\lambda = 100$. We set $K = \frac{T}{4}$ (the parameter in Table 1) as in [13, Sec. 5].

Fig. 7 highlights the different asymptotic tradeoffs summarized in Table 1. On the prover side, Veloz$_{\text{Lin}}$ benefits from its $O(\frac{N}{M})$ proving complexity, whereas the RS-code-based schemes incur additional logarithmic encoding overhead. This distinction becomes increasingly visible as N grows. Specifically, at $N = 2^{30}$, Veloz$_{\text{Lin}}$ proves in 26.9s, compared with 74.3s for DePIP$_{\text{FRI}}$, while Veloz$_{\text{RS}}$, with complexity $O(\frac{N}{M} \log \frac{N}{\log N})$, takes 61.8s. HyperFond and FRIttata exhaust memory before reaching the largest instance, further illustrating the practical advantage of distributing both computation and memory.

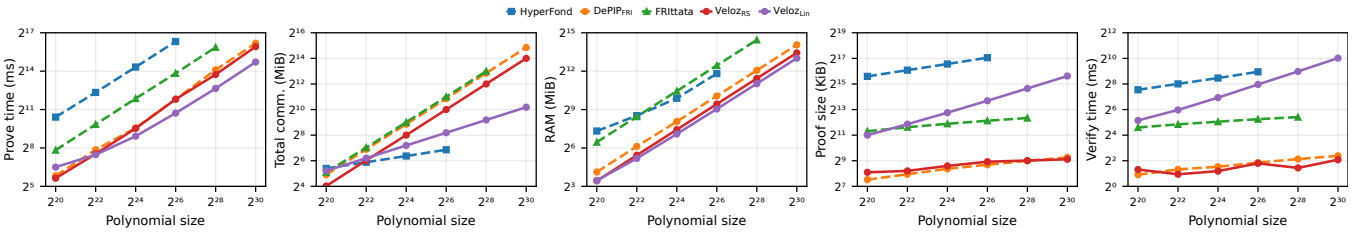

Figure 7: Comparison of Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ to FRIttata, HyperFond, and $\text{DePIP}_{\text{FRI}}$ with $M = 8$ sub-provers

Trends of the verifier side reflect a different tradeoff. With M fixed, the proof size and verification complexity of prior schemes are essentially $O(\log^2 N)$, while those of Veloz_{RS} are $O(\frac{\log^2 N}{\log \log N})$. This difference becomes visible at larger scales in proof size: the curves of Veloz_{RS} and $\text{DePIP}_{\text{FRI}}$ cross between $N = 2^{28}$ and 2^{30}, and at $N = 2^{30}$ their proofs are 554 KB and 613 KB, respectively. The crossover is consistent with the additional $\frac{1}{\log \log N}$ factor of Veloz_{RS}, whose benefit becomes noticeable as N increases. The verification time of Veloz_{RS} also displays a mild "zigzag" across different N, which again results from our setting for μ_k, as explained in Sec. 5.1. Meanwhile, $\text{Veloz}_{\text{Lin}}$ trades larger proofs and slower verification for linear proving time and efficient communication.

Overall, Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ enjoy fast proof generation, which benefits from our distribution design with sublinear communication and efficient memory usage. While Veloz_{RS} favors compact proofs and fast verification, $\text{Veloz}_{\text{Lin}}$ exhibits efficient communication. Both demonstrate considerable improvement over prior schemes.

Remark. Following prior distributed PCS evaluations [13], [15], we simulate sub-provers on a single cloud server in our experiments. In a physically distributed deployment, network latency may introduce additional overhead. Particularly, to prevent $\mathcal{P}_0$ from becoming a bottleneck as M and the number of SumCheck rounds grow, our buffered SumCheck reduces its aggregation workload from $O(M)$ to $O(\log M)$, at the cost of $\log M - 1$ additional sequential synchronization stages per SumCheck round, compared with previous distribution methods. That said, this synchronization overhead is small compared with the overall proving time. For example, with $M = 16$ and $N = 2^{30}$, Veloz_{RS} invokes approximately $\log N - \lceil \log |I_{\ell-1}| \rceil = 30 - \lceil \log 148 \rceil = 22$ SumCheck rounds, resulting in 66 additional synchronization stages. Assuming 1ms LAN latency per synchronization, this contributes about 0.066s, which is minor compared with the measured 40.6s proving time.

6. Further Discussion

Choice of instantiations. The two instantiations of Veloz provide different tradeoffs, and apply to various application scenarios. Specifically, Veloz_{RS} is preferable when proof size and verifier efficiency are critical, such as in blockchain scaling and publicly verifiable SNARK, where proofs may be transmitted to and verified by resource-constrained users. In contrast, $\text{Veloz}_{\text{Lin}}$ is more suitable when prover efficiency or field flexibility is prioritized. Examples include proving machine learning inference, where maximizing proving throughput is important, and settings where Veloz is embedded into a larger proof system with a predetermined field, as $\text{Veloz}_{\text{Lin}}$ does not require an FFT-friendly field.

Furthermore, Veloz is not restricted to RS or Brakedown codes, as the underlying row-wise encoding can be instantiated independently. For example, the recent Quasi-Abelian (QA) code [46] is an interesting candidate. Compared to Brakedown with matched code rate and security parameter, QA code enjoys higher minimum distance and a more efficient encoding algorithm, which can potentially reduce query number and thus proof size, and simultaneously benefit prover efficiency. Instantiating the framework with other efficient linear codes as QA code is therefore a promising direction for improving concrete performance.

Scope of Veloz. Our framework targets the row-separable encoding and interleaved-code-based proximity test, as characterized by the distribution interface in Sec. 1.2. In contrast, schemes such as Orion [7], Blaze [40], Ligero++ [47], and BrakingBase [48] obtain shorter proofs through composition instead of iteration. Therefore, while their inner Ligero-based IOPP may admit our distribution methods, the current framework does not directly applies to the outer composition layer. Extending the aggregation mechanism to these constructions would require additional techniques, which is an interesting direction for future research.

7. Conclusion

We introduced Veloz, a novel distribution framework for interleaved-code-based multilinear PCS agnostic to the concrete linear code, which *first* achieves communication sublinear in the size of polynomial, and sub-prover number-independent proof size. We instantiate Veloz with RS code and Brakedown code [8], denoted as Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$, respectively. With both featuring transparent setup and plausible quantum resilience, Veloz_{RS} exhibits efficient verification and tiny proof size, while $\text{Veloz}_{\text{Lin}}$ enjoys linear proving time and field agnosticism. Experimental results demonstrate their remarkable scalability.

References

[1] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology - ASIACRYPT 2010* (M. Abe, ed.), (Berlin, Heidelberg), pp. 177–194, Springer Berlin Heidelberg, 2010.

[2] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography Conference*, pp. 222–242, Springer, 2013.

[3] J. Lee, "Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments," in *Theory of Cryptography* (K. Nissim and B. Waters, eds.), (Cham), pp. 1–34, Springer International Publishing, 2021.

[4] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 2087–2104, Association for Computing Machinery, 2017.

[5] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast Reed-Solomon Interactive Oracle Proofs of Proximity," in *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)* (I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, eds.), vol. 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 14:1–14:17, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

[6] E. Ben-Sasson, L. Goldberg, S. Kopparty, and S. Saraf, "DEEP-FRI: Sampling outside the box improves soundness." Cryptology ePrint Archive, Paper 2019/336, 2019.

[7] T. Xie, Y. Zhang, and D. 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*, (Berlin, Heidelberg), p. 299–328, Springer-Verlag, 2022.

[8] A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-time and field-agnostic snarks for r1cs," in *Advances in Cryptology - CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, (Berlin, Heidelberg), pp. 193–226, Springer-Verlag, 2023.

[9] H. Zeilberger, B. Chen, and B. Fisch, "Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes," in *Advances in Cryptology – CRYPTO 2024* (L. Reyzin and D. Stebila, eds.), (Cham), pp. 138–169, Springer Nature Switzerland, 2024.

[10] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, "Stir: Reed-solomon proximity testing with fewer queries," in *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part X*, (Berlin, Heidelberg), p. 380–413, Springer-Verlag, 2024.

[11] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, "Whir: Reed–solomon proximity testing with super-fast verification," in *Advances in Cryptology – EUROCRYPT 2025* (S. Fehr and P.-A. Fouque, eds.), (Cham), pp. 214–243, Springer Nature Switzerland, 2025.

[12] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, (New York, NY, USA), p. 3003–3017, Association for Computing Machinery, 2022.

[13] H. Xu, M. Gama, E. H. Beni, and J. Kang, "FRIttata: A FRI-based polynomial commitment scheme for distributed proof generation," *IACR Communications in Cryptology*, vol. 2, no. 4, 2026.

[14] Y. Yu, M. Liu, Y. Zhang, T. Ma, S.-F. Sun, M. H. Au, and D. Gu, "Hyperfond: A transparent and post-quantum distributed snark with polylogarithmic communication," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, ASIA CCS '26, (New York, NY, USA), p. 869–885, Association for Computing Machinery, 2026.

[15] W. Li, Z. Zhang, S. S. M. Chow, Y. Guo, B. Gao, X. Song, Y. Deng, and J. Liu, "Shred-to-Shine metamorphosis of (distributed) polynomial commitments," in *35th USENIX Security Symposium (USENIX Security 26)*, (Baltimore, MD), pp. 5889–5908, USENIX Association, Aug. 2026.

[16] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles," *J. ACM*, vol. 62, Sept. 2015.

[17] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity." Cryptology ePrint Archive, Paper 2018/046, 2018.

[18] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Advances in Cryptology – CRYPTO 2019* (A. Boldyreva and D. Micciancio, eds.), (Cham), pp. 733–764, Springer International Publishing, 2019.

[19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge." Cryptology ePrint Archive, Paper 2019/953, 2019.

[20] B. Bünz, B. Fisch, and A. 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*, (Berlin, Heidelberg), p. 677–706, Springer-Verlag, 2020.

[21] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Advances in Cryptology – CRYPTO 2020* (D. Micciancio and T. Ristenpart, eds.), (Cham), pp. 704–737, Springer International Publishing, 2020.

[22] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Advances in Cryptology – EUROCRYPT 2020* (A. Canteaut and Y. Ishai, eds.), (Cham), pp. 738–768, Springer International Publishing, 2020.

[23] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "Hyperplonk: Plonk with linear-time prover and high-degree custom gates," in *Advances in Cryptology – EUROCRYPT 2023* (C. Hazay and M. Stam, eds.), (Cham), pp. 499–530, Springer Nature Switzerland, 2023.

[24] H. S. Bowe, T. G. Hornby, and N. Wilcox, "Zcash protocol specication," 2016.

[25] ethereum foundation, "zk-rollups: Scaling solutions for ethereum."

[26] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "Zkml: An optimizing system for ml inference in zero-knowledge proofs," in *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, (New York, NY, USA), p. 560–574, Association for Computing Machinery, 2024.

[27] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica, "Dizk: A distributed zero knowledge proof system," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 675–692, USENIX Association, Aug. 2018.

[28] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang, "Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 1777–1793, 2024.

[29] M. Rosenberg, T. Mopuri, H. Hafezi, I. Miers, and P. Mishra, "Hekaton: Horizontally-scalable zksnarks via proof aggregation," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, (New York, NY, USA), p. 929–940, Association for Computing Machinery, 2024.

[30] C. Li, P. Zhu, Y. Li, C. Hong, W. Qu, and J. Zhang, " HyperPianist: Pianist with Linear-Time Prover and Logarithmic Communication Cost ," in *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 3383–3401, IEEE Computer Society, 2025.

[31] W. Wang, F. Shi, D. Vilardell, and F. Zhang, "Cirrus: Performant and accountable distributed SNARK," in *33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026*, The Internet Society, 2026.

[32] W. Li, Z. Zhang, Y. Li, P. Zhu, C. Hong, and J. Liu, "Soloist: Distributed snark for r1cs with constant proof size," in *Advances in Cryptology – EUROCRYPT 2026: 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10–14, 2026, Proceedings, Part VII*, (Berlin, Heidelberg), p. 63–94, Springer-Verlag, 2026.

[33] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[34] A. Novakovic and G. Angeris, "Ligerito: A small and concretely fast polynomial commitment scheme." Cryptology ePrint Archive, Paper 2025/1187, 2025.

[35] O. Reingold, G. N. Rothblum, and R. D. Rothblum, "Constant-round interactive proofs for delegating computation," in *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, (New York, NY, USA), p. 49–62, Association for Computing Machinery, 2016.

[36] E. Ben-Sasson, A. Chiesa, A. Gabizon, M. Riabzev, and N. Spooner, "Interactive Oracle Proofs with Constant Rate and Query Complexity," in *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)* (I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl, eds.), vol. 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 40:1–40:15, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

[37] J. Bootle, A. Chiesa, and J. Groth, "Linear-time arguments with sublinear verification from tensor codes," in *Theory of Cryptography* (R. Pass and K. Pietrzak, eds.), (Cham), pp. 19–46, Springer International Publishing, 2020.

[38] B. E. Diamond and J. Posen, "Proximity testing with logarithmic randomness," *IACR Commun. Cryptol.*, vol. 1, no. 1, 2024.

[39] Y. Guo, X. Liu, K. Huang, W. Qu, T. Tao, and J. Zhang, "Deepfold: efficient multilinear polynomial commitment from reed-solomon code and its application to zero-knowledge proofs," in *Proceedings of the 34th USENIX Conference on Security Symposium*, SEC '25, (USA), USENIX Association, 2025.

[40] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger, "Blaze: Fast snarks from interleaved raa codes," in *Advances in Cryptology – EUROCRYPT 2025* (S. Fehr and P.-A. Fouque, eds.), (Cham), pp. 123–152, Springer Nature Switzerland, 2025.

[41] N. Ron-Zewi and R. Rothblum, "Local proofs approaching the witness length," *J. ACM*, vol. 71, June 2024.

[42] Y. Guo, H. Lou, W. Qu, Z. Xu, and J. Zhang, "LigeSIS: Distribution-friendly polynomial commitment based on error-correcting code." Cryptology ePrint Archive, Paper 2026/751, 2026.

[43] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *J. ACM*, vol. 39, p. 859–868, oct 1992.

[44] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86* (A. M. Odlyzko, ed.), (Berlin, Heidelberg), pp. 186–194, Springer Berlin Heidelberg, 1987.

[45] arkworks contributors, "`arkworks` zksnark ecosystem," 2022.

[46] Z. Li, H. Liu, C. Xing, Y. Yao, and C. Yuan, "More efficient SNARKs via quasi-abelian codes: Faster, smaller, and field-agnostic." Cryptology ePrint Archive, Paper 2026/939, 2026.

[47] R. Bhadauria, Z. Fang, C. Hazay, M. Venkitasubramaniam, T. Xie, and Y. Zhang, "Ligero++: A new optimized sublinear iop," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, (New York, NY, USA), p. 2025–2038, Association for Computing Machinery, 2020.

[48] V. Nair, A. Sharma, and B. Thankey, "BrakingBase - a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme." Cryptology ePrint Archive, Paper 2024/1825, 2024.

[49] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," in *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986*, (Berlin, Heidelberg), p. 31–60, Springer-Verlag, 2016.

Appendix A.
Additional Backgrounds on Proofs and Arguments

PoK and ARK. We adapt the definitions of proof and argument of knowledge (PoK, ARK) from [23]. Recall that an *indexed* relation $\mathcal{R}$ is a set of triples $(\mathtt{i}, \mathbb{x}; \mathbb{w})$, where the index $\mathtt{i}$ is fixed in setup phase. A language $\mathcal{L}(\mathcal{R})$ corresponds to $\mathcal{R}$ such that $(\mathtt{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R}) \Leftrightarrow (\mathtt{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$.

Definition 2 (PoK and ARK [23])**.** *An argument of knowledge $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ for an indexed relation $\mathcal{R}$ runs as follows. Given an index $\mathtt{i}$, common input $\mathbb{x}$ and prover witness $\mathbb{w}$, the deterministic indexer $\mathcal{I}$ outputs $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathtt{i})$. The output of the verifier is denoted by the random variable $\langle \mathcal{P}(\mathsf{gp}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle$.*

Π is (perfectly) complete if for all $(\mathtt{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\begin{array}{c|c} \langle \mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \\ \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle = 1 \end{array} \middle| \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathtt{i}) \end{array}\right] = 1.$$

Π is (computationally) sound if for every pair of probabilistic polynomial time (PPT) adversarial algorithm $(\mathcal{A}_1, \mathcal{A}_2)$, the following probability is $\mathsf{negl}(\lambda)$

$$\Pr\left[\begin{array}{c|c} \langle \mathcal{A}_2(\mathtt{i}, \mathbb{x}, \mathsf{st}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle \\ = 1 \wedge (\mathtt{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}) \end{array} \middle| \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathtt{i}, \mathbb{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathtt{i}) \end{array}\right].$$

If this probability is negligible for unbounded $(\mathcal{A}_1, \mathcal{A}_2)$, then Π is statistically sound.

Π is further knowledge sound if for any pair of PPT adversarial algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a probabilistic expected-polynomial-time extractor $\mathcal{E}$ such that given oracle access to $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$

$$\Pr\left[\begin{array}{c|c} \langle \mathcal{A}_2(\mathtt{i}, \mathbb{x}, \mathsf{st}), \mathcal{V}(\mathsf{vp}, \mathbb{x}) \rangle \\ = 1 \wedge (\mathtt{i}, \mathbb{x}; \mathbb{w}) \notin \mathcal{R} \end{array} \middle| \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathtt{i}, \mathbb{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathtt{i}) \\ \mathbb{w} \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{gp}, \mathtt{i}, \mathbb{x}) \end{array}\right].$$

If $\mathcal{A}$ is unbounded, Π is called a proof of knowledge.

Π is public-coin if all messages from $\mathcal{V}$ can be computed as a deterministic function given a random public input.

SNARK. An ARK is succinct if proof size is $\mathsf{poly}(\lambda, o|\mathbb{w}|)$ and verification is $\mathsf{poly}(\lambda, |\mathbb{x}|, o|\mathbb{w}|)$. A public-coin ARK can be made non-interactive via Fiat-Shamir heuristic [44]. It is called a SNARK if further satisfying succinctness.

IOP. We briefly recall an information-theoretic proof system called interactive oracle proof (IOP), where the verifier possesses oracle access to prover messages.

Definition 3 (Interactive Oracle Proof (IOP) [49])**.** *An ℓ-round public-coin interactive oracle proof $\mathsf{IOP} = (\mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ runs as follows. Initially $\mathcal{P}$ sends an oracle string π_0. In round $k \in [1 : \ell+1]$, the verifier $\mathcal{V}$ samples and sends a random challenge α_k, and the prover replies with an oracle string π_k. After k rounds of interaction the verifier queries some entries of the oracle strings $(\pi_0, \cdots, \pi_\ell)$ and outputs a bit b.*

IOP *is complete if for every* $(\mathbb{x}; \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\begin{array}{c|c} \mathcal{V}^{(\pi_0,\cdots,\pi_\ell)}(\mathbb{x}, & \pi_0 \leftarrow \mathcal{P}(\mathbb{x}; \mathbb{w}) \\ \alpha_1, \cdots, \alpha_\ell) = 1 & \pi_1 \leftarrow \mathcal{P}(\mathbb{x}, \alpha_1; \mathbb{w}) \\ & \cdots \\ & \pi_\ell \leftarrow \mathcal{P}(\mathbb{x}, \alpha_1, \cdots, \alpha_\ell; \mathbb{w}) \end{array}\right] = 1,$$

taken over $\{\alpha_k \leftarrow_{\$} \mathbb{F}\}_{k=1}^{\ell}$ *(similarly hereinafter).*

IOP *is sound if for any* $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$ *and any unbounded* $\mathcal{A}$*, the following probability is* $\mathsf{negl}(\lambda)$

$$\Pr\left[\begin{array}{c|c} \mathcal{V}^{(\pi_0,\cdots,\pi_\ell)}(\mathbb{x}, & \pi_0 \leftarrow \mathcal{A}(\mathbb{x}) \\ \alpha_1, \cdots, \alpha_\ell) = 1 & \pi_1 \leftarrow \mathcal{A}(\mathbb{x}, \alpha_1) \\ & \cdots \\ & \pi_\ell \leftarrow \mathcal{A}(\mathbb{x}, \alpha_1, \cdots, \alpha_\ell) \end{array}\right].$$

IOP of proximity. An IOP of proximity (IOPP) is similar to IOP except that the witness $\mathbb{w}$ is also sent as an oracle, and the verifier can only query $q \ll |\mathbb{w}|$ entries of $\mathbb{w}$.

Definition 4 (Interactive Oracle Proof of Proximity (IOPP) [9])**.** *A public-coin IOP* $(\mathcal{P}, \mathcal{V})$ *for relation* $\mathcal{R}$ *is an IOP of Proximity if it satisfies IOP completeness and* $\beta(\cdot)$*-IOPP soundness: for every* $(\mathbb{x}; \mathbb{w})$ *where* $\mathbb{w}$ *is* δ*-far (in relative Hamming distance) from any* $\mathbb{w}'$ *so* $(\mathbb{x}; \mathbb{w}') \in \mathcal{R}$*, it holds that for any unbounded* $\mathcal{A}$*, the following probability is upper bounded by* $\beta(\delta)$

$$\Pr\left[\begin{array}{c|c} \mathcal{V}^{(\pi_0,\cdots,\pi_\ell)}(\mathbb{x}, & \pi_0 \leftarrow \mathcal{A}(\mathbb{x}; \mathbb{w}) \\ \alpha_1, \cdots, \alpha_\ell) = 1 & \pi_1 \leftarrow \mathcal{A}(\mathbb{x}, \alpha_1; \mathbb{w}) \\ & \cdots \\ & \pi_\ell \leftarrow \mathcal{A}(\mathbb{x}, \alpha_1, \cdots, \alpha_\ell; \mathbb{w}) \end{array}\right].$$

Appendix B. Proof of Theorem 1

We now prove completeness, binding and extractability of our construction.

Proof. **Completeness**: we reduce completeness to that in Ligerito [34]. As in Sec. 3, evaluation consists of the distributed partial SumCheck and the distributed vector-matrix multiplication. For partial SumCheck, a single prover $\mathcal{P}$ in round k sends the verifier $\mathcal{V}$ $s_k(x_k)$, which in distribution is $\sum_{i=0}^{M-1} s_k^{(i)}(x_k)$, with $s_k^{(i)}(x_k)$ locally calculated by $\mathcal{P}_i$. As distribution only alters how $s_k(x_k)$ is obtained, completeness is unaffected. On the other hand, vector-matrix multiplication consists of the following three phases:
(1) Distributed encoding: as in Ligero [4] and Ligerito [34], $\mathbf{U}$ is encoded row-wise. In our distribution, we delegate such encodings to several sub-provers, without modifying how each row is encoded. Therefore, completeness is preserved.
(2) Distributed commitment and querying: in this phase we aim to test whether the encoding $\hat{\mathbf{U}}$ is close to the code space $\mathcal{C}$. The way $\mathcal{V}$ queries for $(\hat{\mathbf{U}})_I$ is distribution-agnostic, since our distribution mandates that $\mathcal{P}_0$ concatenate the columns $(\hat{\mathbf{U}}^{(i)})_I$ from other sub-provers to obtain and send $(\hat{\mathbf{U}})_I$, which is identical to what a single prover $\mathcal{P}$ would send.

As for computing $\mathsf{MC}(\hat{\mathbf{U}})$, in the single-prover setting, $\mathcal{P}$ directly hashes $\hat{\mathbf{U}}$ column-wise, obtains a vector of hash values $\mathbf{h}$, and builds a Merkle tree with roots being $\mathbf{h}$. In our distribution, columns are hashed twice, i.e., $\hat{\mathbf{U}} \to \{\mathbf{h}^{(i)}\}_{i=0}^{M-1} \to \mathbf{h}$, before Merkle tree construction. Therefore, we mandate that $\mathcal{V}$ check $(\hat{\mathbf{U}})_I$ according to the way $\mathbf{h}_I$ is obtained, to preserve completeness.
(3) Distributed linear combination: by linearity, we have $\mathbf{u} = \mathbf{r}_{\otimes} \cdot \mathbf{U} = \sum_{i=0}^{M-1} \mathbf{r}_{\otimes}^{(i)} \cdot \mathbf{U}^{(i)}$, so completeness is clear.
Binding: (adapted from [38, Sec. 4.2]) assume that, given $\mathsf{gp} = (M, \ell, \{\mu_k', \mu_k, \mathbf{G}_k\}_{k=1}^{\ell-1})$, a PPT adversary $\mathcal{A}$ outputs a commitment cm for $f_0 \neq f_1 \in \mathcal{F}_{\mu}^{(\leq 1)}$ with partial opening hints t_0 and t_1, respectively, such that $\mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f_i, t_i) = 1$ for $i = 0, 1$. Let $\mathcal{M}_i$ denote the set of columns missing from t_i, and denote $\mathbf{Mat}_{2^{\mu_1'} \times 2^{\mu_1}}(\mathbf{f}_i)$ by $\mathbf{M}_i$. Define $\mathcal{X}_i := \Delta_{\mathcal{C}_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_i), t_i) \cup \mathcal{M}_i$ for $i = 0, 1$. As both openings are accepted, $|\mathcal{X}_i| < \frac{d_1}{2}$ for $i = 0, 1$. Hence, for their union $\mathcal{X} := \mathcal{X}_0 \cup \mathcal{X}_1$, it holds that $|\mathcal{X}| < d_1$.

We next show that, except with negligible probability, $\Delta_{\mathcal{C}_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_0), \mathsf{Enc}_1(\mathbf{M}_1)) \subseteq \mathcal{X}$. To this end, consider some $j \notin \mathcal{X}$. Now that $j \notin \mathcal{M}_0 \cup \mathcal{M}_1$, both t_0 and t_1 contain valid Merkle openings at index j against the same commitment cm. If $t_0[\cdot, j] \neq t_1[\cdot, j]$, then $\mathcal{A}$ locates a hash collision, which is of negligible probability. Therefore, overwhelmingly, the two opened columns are identical. Moreover, since $j \notin \Delta_{\mathcal{C}_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_i), t_i)$ for $i = 0, 1$, each opened column agrees with the corresponding column of $\mathsf{Enc}_1(\mathbf{M}_i)$. Hence, $\mathsf{Enc}_1(\mathbf{M}_0)[\cdot, j] = \mathsf{Enc}_1(\mathbf{M}_1)[\cdot, j]$. Consequently, the desired inclusion holds, so $|\Delta_{\mathcal{C}_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_0), \mathsf{Enc}_1(\mathbf{M}_1))| < d_1$.

On the other hand, since $f_0 \neq f_1$, we have $\mathbf{M}_0 \neq \mathbf{M}_1$, so there exists a row index p such that $\mathbf{M}_0[p, \cdot] \neq \mathbf{M}_1[p, \cdot]$. Therefore, $\mathsf{Enc}_1(\mathbf{M}_0[p, \cdot])$ and $\mathsf{Enc}_1(\mathbf{M}_1[p, \cdot])$ differ at least d_1 positions, yielding $|\Delta_{\mathcal{C}_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_0), \mathsf{Enc}_1(\mathbf{M}_1))| \geq d_1$, a contradiction. Hence, we conclude that $f_0 = f_1$.
Soundness: (adapted from [34, Sec. 6.3]) the soundness error in round $1 \leq k \leq \ell - 1$ arises from (1) a μ_k'-round distributed partial SumCheck (Protocol 1), (2) a distributed commitment and querying protocol (Protocol 2) for the matrix $\mathbf{U}_k \in \mathbb{F}^{2^{\mu_k'} \times 2^{\mu_k}}$, and (3) batching $|I_k| + 1$ columns with the random vector $\mathbf{s}_k = (1, \gamma_k, \cdots, \gamma_k^{|I_k|})$ (Protocol 4).

For (1), by the well-known Schwartz-Zippel Lemma, the probability that a random linear polynomial $s^*(x)$ satisfies a linear constraint is $\frac{1}{|\mathbb{F}|}$, so a μ_k'-round partial SumCheck renders up to $\frac{\mu_k'}{|\mathbb{F}|}$ error. For (2), the probability that $\hat{\mathbf{U}}^*$ is not an interleaved codeword of $\mathbf{U}_k$ yet $\mathcal{V}$ accepts after querying $|I_k|$ columns is $(1 - \frac{2^{\mu_k} - \rho \cdot 2^{\mu_k} + \rho}{2 \cdot 2^{\mu_k}})^{|I_k|}$ for RS code, or $(1 - \frac{\rho \cdot d}{3 \cdot 2^{\mu_k}})^{|I_k|}$ for a $[\frac{2^{\mu_k}}{\rho}, 2^{\mu_k}, d_k]$-linear code from Brakedown [8]. The probability that $\mathbf{u}^* \neq \mathbf{u}_k$ yet $(\mathbf{r}_k)_{\otimes} \cdot \hat{\mathbf{U}}_k = u^*(\mathbf{r}_k)$ is $\frac{\frac{2^{\mu_k}}{\rho} \cdot \mu_k}{|\mathbb{F}|}$ in $\mathsf{Veloz}_{\mathsf{RS}}$, or $\frac{d \cdot \mu_k}{3|\mathbb{F}|}$ in $\mathsf{Veloz}_{\mathsf{Lin}}$, where u^* is the multilinear polynomial interpolated from $\mathbf{u}^*$. For (3), again by the Schwartz-Zippel Lemma, the error is $\frac{|I_k|+1}{|\mathbb{F}|}$. Thus, conditioned on $\log|\mathbb{F}| = \Omega(\lambda)$, the error is $\epsilon_{\mathsf{RS}} = O\Big(\sum_{k=1}^{\ell-1} \big[(1 - \frac{2^{\mu_k} - \rho \cdot 2^{\mu_k} + \rho}{2^{1+\mu_k}})^{|I_k|} + \frac{2^{\mu_k} \cdot \mu_k}{\rho|\mathbb{F}|}\big]\Big)$, and $\epsilon_{\mathsf{Lin}} = O\Big(\sum_{k=1}^{\ell-1} \big[(1 - \frac{\rho \cdot d_k}{3 \cdot 2^{\mu_k}})^{|I_k|} + \frac{d_k \cdot \mu_k}{3|\mathbb{F}|}\big]\Big)$, respectively.
Extractability: we prove extractability in the random oracle

model, following the extractor constructed in the proof of [38, Theorem 5]. After a commitment c is fixed, their extractor consists of a TREEBUILDER procedure and a rewinding-based extraction procedure. The former constructs from the adversary's random oracle queries a partial opening hint $\hat{\mathbf{U}}'$, possibly with a set $\mathcal{M}$ of missing columns, and the latter repeatedly rewinds $\mathcal{A}$ to the state immediately after c, to obtain accepting row-wise linear combinations at fresh random evaluation points. By the extraction analysis of [38, Theorem 5], overwhelmingly, these accepting row combinations determine a matrix $\mathbf{U}'$ such that $|\Delta_{\mathcal{C}}^{\mu'}(\mathsf{Enc}(\mathbf{U}'), \hat{\mathbf{U}}') \cup \mathcal{M}| < \frac{d}{2}$. Hence, $\hat{\mathbf{U}}'$ and the polynomial corresponding to $\mathbf{U}'$ satisfy the opening predicate. Let Open_k denote this predicate instantiated with $(\mu'_k, \mu_k, \mathcal{C}_k)$, so that $\mathsf{Open}_1 = \mathsf{Open}$. In the following, we define $\ell - 1$ extractors $\{\mathcal{E}_k\}_{k=1}^{\ell-1}$ and apply this extraction argument recursively to the iterations.

We prove the recursive extraction by backward induction on k. In particular, for every accepting instance at level k, $\mathcal{E}_k$ extracts a matrix $\mathbf{U}'_k$ and a partial opening hint $\hat{\mathbf{U}}'_k$ with overwhelming probability, such that $y_k = \langle \mathbf{u}'_{k-1}, \mathbf{z}_k \rangle$ and $\mathsf{Open}_k(\mathsf{gp}, \mathsf{cm}_k; f'_k, \hat{\mathbf{U}}'_k) = 1$, where $\mathbf{u}'_{k-1}$ is $\mathbf{U}'_k$ flattened, and f'_k is the multilinear polynomial corresponding to $\mathbf{U}'_k$.

We start from $k = \ell - 1$. Fix the commitment $\mathsf{cm}_{\ell-1}$, and let $\mathcal{E}_{\ell-1}$ run TREEBUILDER once to construct a partial opening hint $\hat{\mathbf{U}}'_{\ell-1}$ with missing-column set $\mathcal{M}_{\ell-1}$. For $j = 0$, $\mathcal{E}_{\ell-1}$ uses the original accepting continuation and obtains $\mathbf{u}'_{\ell-1,0} := \mathbf{u}'_{\ell-1}$, with $\mathbf{r}_{\ell-1,0} := \mathbf{r}_{\ell-1}$ and $I_{\ell-1,0} := I_{\ell-1}$. For each $j \in [1 : 2^{\mu'_{\ell-1}}]$, it rewinds $\mathcal{A}$ to the state immediately after $\mathsf{cm}_{\ell-1}$, reprograms the random oracle for fresh evaluation randomness $\boldsymbol{\rho}_{\ell-1,j} := \mathbf{r}_{\ell-1,j} \| \mathbf{r}_{\ell,j}$, obtains $\mathbf{u}'_{\ell-1,j}$ before freshly sampling $I_{\ell-1,j}$, and continues the branch until acceptance. It also sets $s_{\ell-1,j} := \langle \mathbf{u}'_{\ell-1,j}, (\mathbf{r}_{\ell,j})_{\otimes} \rangle$.

By the extraction analysis of [38, Theorem 5], overwhelmingly, the $2^{\mu'_{\ell-1}}$ accepting row combinations satisfy $\mathbf{u}'_{\ell-1,j} = (\mathbf{r}_{\ell-1,j})_{\otimes} \cdot \mathbf{U}'_{\ell-1}$ for a single matrix $\mathbf{U}'_{\ell-1}$ for all j, and $\mathsf{Open}_{\ell-1}(\mathsf{gp}, \mathsf{cm}_{\ell-1}; f'_{\ell-1}, \hat{\mathbf{U}}'_{\ell-1}) = 1$. Indeed, by [38, Lemma 9], the matrix with rows being $(\mathbf{r}_{\ell-1,j})_{\otimes}$ is invertible except with negligible probability, so the collected relations uniquely determine $\mathbf{U}'_{\ell-1}$. Moreover, on every accepting continuation, the final verification in Protocol 4 directly enforces the required queried-column consistency and $y'_{\ell-1,j} = \langle \mathbf{u}'_{\ell-1,j}, \mathbf{w}_{\ell-1,j} \rangle$. On the original continuation $j = 0$, as $\mathbf{u}'_{\ell-1,0} = (\mathbf{r}_{\ell-1,0})_{\otimes} \cdot \mathbf{U}'_{\ell-1}$, partial SumCheck soundness implies, overwhelmingly, $y_{\ell-1} = \langle \mathbf{u}'_{\ell-2}, \mathbf{z}_{\ell-1} \rangle$. Hence, the induction statement holds for $k = \ell - 1$.

For $1 \le k \le \ell - 2$, assume inductively that $\mathcal{E}_{k+1}$ satisfies the induction statement above. Fix cm_k and let $\mathcal{E}_k$ run TREEBUILDER to construct a partial opening hint $\hat{\mathbf{U}}'_k$ with missing-column set $\mathcal{M}_k$. For $j = 0$, it follows the original accepting continuation; for each $j \in [1 : 2^{\mu'_k}]$, it rewinds $\mathcal{A}$ to the state immediately after cm_k and reprograms the random oracle to obtain fresh $\mathbf{r}_{k,j}$. By Protocol 4, $\mathcal{A}$ fixes $\mathsf{cm}_{k+1,j}$ before $I_{k,j}$ and $\gamma_{k,j}$ are sampled. After continuing the branch until acceptance, $\mathcal{E}_k$ invokes $\mathcal{E}_{k+1}$ on the instance at level $k+1$ associated with $\mathsf{cm}_{k+1,j}$ and flattens $\mathbf{U}'_{k+1,j}$ as $\mathbf{u}'_{k,j}$. By the induction hypothesis, $y_{k+1,j} = \langle \mathbf{u}'_{k,j}, \mathbf{z}_{k+1,j} \rangle$ and $\mathsf{Open}_{k+1}(\mathsf{gp}, \mathsf{cm}_{k+1,j}; f'_{k+1,j}, \hat{\mathbf{U}}'_{k+1,j}) = 1$.

Next, we show that $\mathbf{u}'_{k,j}$ is also a valid row-combination message for cm_k. Expanding the evaluation above gives $\langle \mathbf{d}_{k,j}, \mathbf{s}_{k,j} \rangle = 0$, where $\mathbf{d}_{k,j} := (\mathbf{v}_{k,j} - \mathbf{u}'_{k,j} \cdot (\mathbf{G}_k)_{I_{k,j}}) \,\|\, (y'_{k,j} - \langle \mathbf{u}'_{k,j}, \mathbf{w}_{k,j} \rangle)$. Since the extracted child opening is valid, binding implies that, overwhelmingly, $\mathbf{u}'_{k,j}$ is uniquely determined by the fixed $\mathsf{cm}_{k+1,j}$. Hence, after $I_{k,j}$ is fixed, $\mathbf{d}_{k,j}$ is determined before $\gamma_{k,j}$ is sampled. If $\mathbf{d}_{k,j} \neq \mathbf{0}$, then $\langle \mathbf{d}_{k,j}, \mathbf{s}_{k,j} \rangle$ is a nonzero polynomial in $\gamma_{k,j}$ of degree at most $|I_{k,j}|$, hence vanishing with probability at most $|I_{k,j}|/|\mathbb{F}|$. Thus, overwhelmingly, $\mathbf{d}_{k,j} = \mathbf{0}$, or $\mathbf{v}_{k,j} = \mathbf{u}'_{k,j} \cdot (\mathbf{G}_k)_{I_{k,j}}$ and $y'_{k,j} = \langle \mathbf{u}'_{k,j}, \mathbf{w}_{k,j} \rangle$. Together with the accepted Merkle openings, these give the column-consistency and evaluation equations required for $\mathbf{u}'_{k,j}$ as an accepting row-combination message for cm_k. Applying the extraction analysis of [38, Theorem 5] to $\{\mathbf{u}'_{k,j}, \mathbf{r}_{k,j}\}_{j=0}^{2^{\mu'_k}-1}$ then uniquely determines $\mathbf{U}'_k$ such that $\mathbf{u}'_{k,j} = (\mathbf{r}_{k,j})_{\otimes} \cdot \mathbf{U}'_k$ for all j and $\mathsf{Open}_k(\mathsf{gp}, \mathsf{cm}_k; f'_k, \hat{\mathbf{U}}'_k) = 1$. For $j = 0$, we further use $y'_{k,0} = \langle \mathbf{u}'_{k,0}, \mathbf{w}_{k,0} \rangle$, which, together with $\mathbf{u}'_{k,0} = (\mathbf{r}_{k,0})_{\otimes} \cdot \mathbf{U}'_k$, is the terminal relation of the level-k partial SumCheck instantiated with the extracted $\mathbf{U}'_k$. Since binding makes $\mathbf{U}'_k$ fixed independently of $\mathbf{r}_{k,0}$, partial SumCheck soundness implies, overwhelmingly, that $y_k = \langle \mathbf{u}'_{k-1}, \mathbf{z}_k \rangle$. This closes the induction.

It remains to analyze the conditioning introduced by repeating each rewind until acceptance. Following the argument of [38, Lemma 7], fix any level-k prefix after cm_k and let ϵ_k denote its continuation acceptance probability. For a fresh continuation from this prefix, let $e_k = \mathsf{negl}(\lambda)$ upper-bound the union of the child-extraction, binding, Merkle-consistency, SumCheck, batching, and linear-independence failures used above. If $\epsilon_k \le \sqrt{e_k}$, the probability that this prefix yields an accepting continuation and is expanded by the extractor is itself negligible. Otherwise, for any such bad event B, $\Pr[B \mid \mathsf{Acc}] = \frac{\Pr[B \wedge \mathsf{Acc}]}{\epsilon_k} < \frac{e_k}{\sqrt{e_k}} = \sqrt{e_k}$. Hence, the accepting continuations returned by the previous procedure satisfy the same properties except with negligible probability. This bound applies to each reached prefix, and a union bound over the polynomially many extracted branches yields an overall negligible error.

Applying the induction down to $k = 1$ yields $\mathbf{u}'_0$, which corresponds to $f' \in \mathcal{F}_{\mu_0}^{(\le 1)}$, together with a partial opening hint $\hat{\mathbf{U}}'_1$ such that $\mathsf{Open}(\mathsf{gp}, \mathsf{cm}_1; f', \hat{\mathbf{U}}'_1) = 1$ and $y_1 = \langle \mathbf{u}'_0, \mathbf{z}_1 \rangle$. As $y_1 = y$ and $\mathbf{z}_1 = \mathbf{z}_{\otimes}$, the latter is exactly $f'(\mathbf{z}) = y$. Thus, considering also the accumulated protocol soundness error ϵ, except with probability $\epsilon + \mathsf{negl}(\lambda)$, we have $f'(\mathbf{z}) = y \wedge \mathsf{Open}(\mathsf{gp}, \mathsf{cm}_1; f', \hat{\mathbf{U}}'_1) = 1$.

Finally, we bound the extraction time. For each level-k extraction node, denote the running time of TREEBUILDER by T_k, which by [38] satisfies $T_k \le \mathsf{poly}(N)$. On the other hand, for a fixed level-k prefix with acceptance probability ϵ_k, the extractor expands this node only if its initial continuation accepts. Conditioned on doing so, collecting the remaining $2^{\mu'_k} - 1$ accepting continuations takes expected $\frac{2^{\mu'_k}-1}{\epsilon_k}$ trials. Hence, the expected work contributed by this expansion is $1 + \epsilon_k \cdot \frac{2^{\mu'_k}-1}{\epsilon_k} = 2^{\mu'_k}$. Note that the

number of accepting continuations produced at level k is at most $\prod_{j=1}^{k-1} 2^{\mu'_j} = 2^{\sum_{j=1}^{k-1}\mu'_j} = 2^{\mu_0-\mu_{k-1}}$. Therefore, the expected time of constructing the extraction tree is bounded by $1+\sum_{k=1}^{\ell-1}(2^{\mu_0-\mu_{k-1}}\cdot T_k + 2^{\mu_0-\mu_{k-1}+\mu'_k}) = O(\ell\cdot(N\cdot\mathsf{poly}(N)+2^{\mu_0})) = \mathsf{poly}(N)$. □

Appendix C. Detailed Complexity Analysis

In the following, we analyze the complexity in detail. We write $|I_k| = \Theta(\lambda)$ for all k for brevity. To avoid abuse of notations, we denote field operations or elements by $\mathbb{F}$, and hashing or hash values by $\mathbb{H}$.

Prover. In round $1 \le k \le \ell-2$, $\mathcal{P}_i$

- encodes $\mathbf{U}_{k+1}$: $O(2^{\mu'_{k+1}-m}\cdot\mathsf{Enc}_{k+1})$ $\mathbb{F}$;
- Merkle commits to $\hat{\mathbf{U}}_{k+1}$: $O(2^{\mu'_{k+1}-m}\cdot\frac{2^{\mu_{k+1}}}{\rho})$ $\mathbb{H}$ for each $\mathcal{P}_i$, and extra $O(2^m)$ $\mathbb{H}$ for $\mathcal{P}_0$;
- runs a μ'_k-round partial SumCheck: $O(2^{\mu'_k-m}+\mu'_k\cdot(m-\lfloor\log(i+1)\rfloor+1))$ $\mathbb{F}$;
- computes $\mathbf{w}_k$ and $\mathbf{u}_k$: $O(2^{\mu'_k-m+\mu_k})$ $\mathbb{F}$.

In total, the cost of each $\mathcal{P}_i$ is $\sum_{k=0}^{\ell-2}O(2^{\mu'_{k+1}-m}\cdot\mathsf{Enc}_{k+1})$ $\mathbb{F}+\sum_{k=0}^{\ell-2}O(2^{\mu'_{k+1}+\mu_{k+1}-m})$ $\mathbb{H}$, where Enc_k is $O(\mu_k 2^{\mu_k})$ for RS code, and $O(2^{\mu_k})$ for Brakedown code [8].

Synchronization. In round $1 \le k \le \ell-1$, sub-provers synchronize states as they

- compute $s_k(x_k)$ in the distributed partial SumCheck: $O(\log M\cdot\mu'_k)$ rounds;
- exchange columns for local hashing: 1 round;
- aggregate the queried $|I_k|$ columns: 1 round;
- receive the verifier-sampled γ_k for random linear combination and aggregate its result: 1 round.

To sum up, the total number of rounds for synchronization is $O(\sum_{k=1}^{\ell-1}(\log M\cdot\mu'_k+1)) = O(\log M\cdot(\mu-\mu_{\ell-1})+\ell)$.

Communication. In round $1 \le k \le \ell-1$, $\mathcal{P}_i$

- exchanges hash values to commit to $\hat{\mathbf{U}}_k$: $O(2^{\mu_k})$ $\mathbb{H}$;
- receives the SumCheck challenge from $\mathcal{P}_0$ and sends $s_k^{(i)}(x_k)$: $O(1)$ $\mathbb{F}$;
- sends Merkle leaves and paths of $(\hat{\mathbf{U}}_k)_{I_k}$ to $\mathcal{P}_0$ in the querying phase of Protocol 2: $O(\lambda\cdot 2^{\mu'_k-m})$ $\mathbb{F}$; and
- exchanges vectors to compute $\mathbf{w}_k$ and $\mathbf{u}_k$: $O(2^{\mu_k})$ $\mathbb{F}$.

In sum, the total communication cost is $O(\sum_{k=1}^{\ell-1}(\lambda\cdot 2^{\mu'_k}+M\cdot 2^{\mu_k}))$ $\mathbb{F}+O(M\cdot\sum_{k=1}^{\ell-1}2^{\mu_k})$ $\mathbb{H}$.

Verifier. In round $1 \le k \le \ell-1$, $\mathcal{V}$

- runs μ'_k rounds of partial SumCheck: $O(\mu'_k)$ $\mathbb{F}$;
- verifies Merkle paths of $(\hat{\mathbf{U}}_k)_{I_k}$: $O(\lambda\cdot(2^{\mu'_k}+\mu_k))$ $\mathbb{H}$;
- computes $\mathbf{w}_k$: $O(\mu'_1+2^{\mu_1})$ $\mathbb{F}$ for $k=1$ and $O(2^{\mu'_k+\mu_k})$ $\mathbb{F}$ for $k\ge 2$ in $\mathsf{Veloz}_{\mathsf{Lin}}$, and $O(\mu'_1)$ $\mathbb{F}$ for $k=1$ and $O(\lambda\cdot(k-1)\mu'_k)$ $\mathbb{F}$ for $k\ge 2$ in $\mathsf{Veloz}_{\mathsf{RS}}$;
- computes $\mathbf{z}_{k+1}$: $O(\lambda\cdot 2^{\mu_k})$ in $\mathsf{Veloz}_{\mathsf{Lin}}$, and 0 in $\mathsf{Veloz}_{\mathsf{RS}}$ as discussed in Sec. 3.4; and
- verifies $\mathbf{u}_{\ell-1}\cdot(\mathbf{G}_{\ell-1})_{I_{\ell-1}} = (\mathbf{r}_{\ell-1})_\otimes\cdot(\hat{\mathbf{U}}_{\ell-1})_{I_{\ell-1}}$, and $y'_{\ell-1} = \langle\mathbf{w}_{\ell-1},\mathbf{u}_{\ell-1}\rangle$: $O(2^{\mu_{\ell-1}})$ $\mathbb{F}$.

To summarize, verification in $\mathsf{Veloz}_{\mathsf{Lin}}$ costs $O(\mu'_1+2^{\mu_1}+\lambda\cdot\sum_{k=1}^{\ell-2}2^{\mu_k}+2^{\mu_{\ell-1}})$ $\mathbb{F}+O(\lambda\cdot\sum_{k=1}^{\ell-1}(2^{\mu'_k}+\mu_k))$ $\mathbb{H}$. In $\mathsf{Veloz}_{\mathsf{RS}}$, it reduces to $O(\mu'_1+\lambda\cdot(\sum_{k=1}^{\ell-1}(k-1)\cdot\mu'_k)+2^{\mu_{\ell-1}})$ $\mathbb{F}+O(\lambda\cdot\sum_{k=1}^{\ell-1}(2^{\mu'_k}+\mu_k))$ $\mathbb{H}$.

Proof size. In round $1 \le k \le \ell-1$, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ send $\mathcal{V}$ three field elements in partial SumCheck, a Merkle root $\mathsf{MC}(\hat{\mathbf{U}}_k)$ and related paths, and columns of $(\hat{\mathbf{U}}_k)_{I_k}$ for $\mathcal{V}$'s query. In the final round, $\mathcal{P}_0$ sends $\mathbf{u}_{\ell-1}$ directly. The total proof size thus amounts to $O(\sum_{k=1}^{\ell-1}(3\mu'_k+\lambda\cdot 2^{\mu'_k})+2^{\mu_{\ell-1}})$ $\mathbb{F}$ + $O(\lambda\cdot\sum_{k=1}^{\ell-1}\mu_k)$ $\mathbb{H}$.

Asymptotics for $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$. Now we asymptotically summarize the complexity of $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$, respectively. Denote the size of f by $N = 2^{\mu_0}$.

• $\mathsf{Veloz}_{\mathsf{RS}}$. For convenience, we set $\mu'_k = \mu'$ identically for each $k\in[1:\ell]$, so $\mu_k = \mu_0 - k\mu' = \log N - k\mu'$. The proof size is dominated by $O(\lambda(\ell 2^{\mu'}+\sum_{k=1}^{\ell-1}(\log N - k\mu'))) = O(\lambda\ell(2^{\mu'}+\log N+\ell\mu'))$; for verification, it is $O(\lambda(\ell^2\mu'+\ell 2^{\mu'}+\sum_{k=1}^{\ell-1}(\log N-k\mu'))) = O(\lambda\ell(\ell\mu'+2^{\mu'}+\log N))$. Therefore, we set, asymptotically, $2^{\mu'} = \log N = \ell\mu'$, which yields $\mu' = \log\log N$ and $\ell = \frac{\log N}{\log\log N}$. Based on these, the prover complexity is $\sum_{k=1}^{\ell-1}O(2^{\mu'-m}\cdot\frac{N}{2^{k\cdot\mu'}}\cdot(\log N-\log 2^{k\cdot\mu'})) = O\big(\frac{N}{M}(\frac{\log^2 N}{\log N-1}-\frac{\log^2 N\log\log N}{(\log N-1)^2})\big) = O(\frac{N}{M}\log\frac{N}{\log N})$; the communication cost amounts to $O\big(M\cdot\sum_{k=1}^{\ell-1}(\lambda\cdot 2^{\mu'-m}+\frac{N}{(2^{\mu'})^k})\big) = O(\lambda\cdot\frac{\log^2 N}{\log\log N}+M\cdot\frac{N}{\log N})$; and the proof size and verification are $O(\lambda\cdot\frac{\log^2 N}{\log\log N})$.

• $\mathsf{Veloz}_{\mathsf{Lin}}$. The main difference is in verification, where the dominant part now is $O(\lambda\cdot(\sum_{k=1}^{\ell-2}2^{\mu_k}+\sum_{k=1}^{\ell-1}2^{\mu'_k}))$. We let $K := 2^{\mu'_1}$ be a free parameter for a moment, and set $\mu'_k = \mu_k = \frac{\mu_{k-1}}{2}$ for $k\ge 2$. It yields $O\big(\lambda(K+\frac{N}{K})\big)$ verification, and $O\big(\lambda(K+\sqrt{N/K})\big)$ proof size. Since verification and proof size reach their minimum when $K=\sqrt{N}$ and $K=\sqrt[3]{N}$, respectively, we set $K\in[\sqrt[3]{N},\sqrt{N}]$, which yields $O(2^{\log K-m}\cdot\frac{N}{K}) = O(\frac{N}{M})$ prover complexity, $O\big(M\cdot\sum_{k=1}^{\ell-1}(\lambda\cdot 2^{\mu'_k-m}+2^{\mu_k})\big) = O\big(\lambda(K+\sqrt{N/K})+M\cdot\frac{N}{K}\big) = O\big(\lambda K+M\cdot\frac{N}{K}\big)$ communication, $O(\lambda\cdot(K+\frac{N}{K}))$ verification, and $O\big(\lambda(K+\sqrt{N/K})\big) = O(\lambda K)$ proof size.