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

Yuanzhuo Yu
Shanghai Jiao Tong University
Shanghai, China
yzyu2000@sjtu.edu.cn

Shi-Feng Sun
Shanghai Jiao Tong University
Shanghai, China
shifeng.sun@sjtu.edu.cn

Yuncong Zhang
Shandong University
Qingdao, China
yuncong@sdu.edu.cn

Chenhua Fan
Xidian University
Xi'an, China
ord1r3hv@gmail.com

Tianyi Ma
Shanghai Jiao Tong University
Shanghai, China
mty021129@sjtu.edu.cn

Dawu Gu
Shanghai Jiao Tong University
Shanghai, China
dwgu@sjtu.edu.cn

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 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 agnosticity.

We also implement both schemes in Rust and conduct a comprehensive performance evaluation. The experimental results demonstrate their linear scalability with increasing M. More specifically, for $N=2^{30}$ and $M=8$, Veloz_{RS} takes 74.8s for proof generation, achieving a 5.18 × speedup compared to running a single prover, while $\text{Veloz}_{\text{Lin}}$ generates a proof in 26.9s and achieves a 7.02 × speedup.

1 Introduction

A *polynomial commitment scheme* (PCS) [25] 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. [25], a large body of research has been conducted over the last two decades [1, 3, 4, 6, 10, 23, 26, 33, 44, 47]. Generally, the existing works can be classified into two categories: pairing-based PCSs [25, 26, 33], which enjoy fast verification, small proof size, and efficient proof aggregation, due to the algebraic homomorphisms rooted in bilinear groups; and code-based PCSs [1, 3, 4, 6, 10, 23, 44, 47], which in contrast feature a transparent setup and plausibly post-quantum security, as they rely merely on one-wayness of hash functions.

In recent years, PCS has become a crucial building block of *succinct non-interactive argument of knowledge* (SNARK) [7, 14, 15, 17, 21, 22, 38, 43], 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 trade-off between efficiency and security has driven its adoption in cryptocurrencies [12], blockchain scaling [20], and machine learning inference [16]. 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 [27–30, 36, 40–42, 45, 46] 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 dispersing 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, 29, 30, 36, 40, 41] 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 [27] and Cirrus [40] 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.

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 [39]. Motivated by these concerns, a line of recent works [28, 42, 45, 46] focus on developing code-based distributed PCSs, which enjoy transparent setup

Table 1: Comparisons of Veloz to prior code-based distributed PCSs. Here N is the size of a bivariate or multilinear 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 $\text{Veloz}_{\text{Lin}}$. "Field-agnostic?" means whether PCS works over any sufficiently large finite field $\mathbb{F}$.

Schemes	Proving	Total communication	Proof size	Verification	Field-agnostic?
deVirgo [42]	$O(T \log T)$	$O(N)$	$O(\lambda \cdot M + \log^2 T)$	$O(\lambda \cdot M + \log^2 T)$	No
FRIttata [45]	$O(T \log T)^\star$	$O(MK + \lambda \cdot M \log \frac{T}{K} \log T)$	$O\Big(\lambda\big(M \log \frac{T}{K} \log T + \log^2 K\big)\Big)$	$O\Big(\lambda\big(M \log \frac{T}{K} \log T + \log^2 K\big)\Big)$	No
HyperFond [46]	$O(T \log T)$	$O(\lambda \cdot M \log^2 T)$	$O(\lambda \cdot M \log^2 T)$	$O(\lambda \cdot M \log^2 T)$	Yes
$\text{DePIP}_{\text{FRI}}$ [28]	$O(T \log T)$	$O(N)$	$O(\lambda \cdot M + \log^2 T)$	$O(\lambda \cdot M + \log^2 T)$	No
Veloz_{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
$\text{Veloz}_{\text{Lin}}$ (**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 [45, Section 4], which originates from an outer protocol therein. We view it as an independent PCS in this table.

and plausibly post-quantum security, as well as implementation-friendliness in engineering. Due to the lack of homomorphic properties over bilinear groups, however, existing code-based 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 trade-off 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 a recent multilinear PCS, Ligerito [32], we introduce Veloz, a distribution framework for code-based multilinear PCS, and instantiate it with Reed-Solomon (RS) code and the fast linear code from Brakedown [23], respectively. Both schemes 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 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, plausibly quantum resilience, communication sublinear in N, and sub-prover number independent proof size; the detailed comparisons are shown in Table 1. In more detail, our main contributions are summarized as follows.

We propose the *first* distribution framework for multilinear PCS, dubbed Veloz, which generates an evaluation proof with size independent of the number of sub-provers, requiring communication cost sublinear in N. To this end, we propose a tailored distribution technique for the SumCheck protocol [31], and an interleaved code-specific proof aggregation method to eliminate the dependence of proof size from sub-prover number.

Then we instantiate Veloz with RS code and the fast linear code from Brakedown [23], and obtain two concretely efficient schemes Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$. Regarding Veloz_{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 an efficient verification algorithm with complexity $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$, which applies even in the single-prover setting. As to $\text{Veloz}_{\text{Lin}}$, it achieves *for the first time* linear proving time, benefiting from the linear encoding complexity of the Brakedown 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 agnosticity* as Brakedown, which enables its application to any sufficiently large finite field, rather than a well-modified multiplicative subgroup of the base field for efficient fast fourier transform (FFT) as required by RS code-based schemes.

Finally, we implement both Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ 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. Particularly, for $N = 2^{30}$ and $M = 8$, Veloz_{RS} generates a proof with 74.8s, achieving a 5.18 × speedup over the single-prover baseline, with a proof size of 487KB, and verification time of 8.39ms. Under the same conditions, $\text{Veloz}_{\text{Lin}}$ performs proof generation in 26.9s and achieves a 7.02 × speedup; its proof length is 49.2MB, with 1.04s for verification. We also compare to FRIttata [45] and HyperFond [46] based on their publicly available source codes under the same environment, where we fix $M = 8$. The results demonstrate that both Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ are faster and more memory efficient for proof generation. In particular, given $N = 2^{28}$, HyperFond already runs out of memory, while Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ are 1.46 × and 3.29 × faster than FRIttata, respectively. We provide more details in Sect. 5.

We remark that we also give the formal security proof of binding and knowledge soundness properties in Sect. 4.1, which applies to the single-prover setting as well and thus implying the binding and knowledge soundness of [32]. Recall that [32] only shows the completeness and soundness analysis. In addition, we provide a more succinct verification algorithm in Sect. 3.4, which achieves $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ complexity instead of $O(N)$ as in [32].

[1] As analyzed in Sect. 4.2, Veloz_{RS} achieves the optimal proof size $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$, conditioned on $M \le O(\log N)$. We note that our general design is not subjected to this constraint, where M is only bounded by the row number of the encoded matrix.

1.2 Overview

In this section, we first briefly review the background of code-based PCS and prior distribution schemes with trade-offs between linear communication and $O(M)$ overhead in proof size, to better understand our motivation. Then we parse the technical challenges encountered during distribution. Finally, we unfold our main ideas.

Motivation. Often, code-based PCSs build upon a special proof system known as *Interactive Oracle Proof of Proximity* (IOPP) [6, 8, 11, 34, 47], 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 $\mathcal{V}$. $\mathcal{V}$ tests that $\mathbf{u}$ is close to some prescribed linear code space 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 degree bound by first encoding $\mathbf{f}$, and invoking an IOPP to prove that the distance of the encoded vector $\hat{\mathbf{f}} = \mathsf{Enc}(\mathbf{f})$ from C is small.

When it comes to distributed schemes [28, 42, 45, 46], 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 C is reduced to M claims for $\{\hat{\mathbf{f}}^{(i)}\}_{i=0}^{M-1}$ from $\hat{C}$, a sub-code space with adapted dimension. Now we arrive at the following dilemma: to prove that $\hat{\mathbf{f}}^{(i)}$ is close to $\hat{C}$ 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 [28, 42], 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 [46]. FRIttata [45] is a combination of these strategies, by "flexibilizing" the moment when sub-provers exchange elements, and obtains a balance between communication and proof size tuned by a parameter $K \in [1, \frac{N}{M}]$. However, its experimental result still reports an asymptotical linear growth of proof size as M increases.

We observe that the $O(M)$ overhead in [45, 46] 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 for us to focus on the *outer* design among these codewords, i.e., how we arrange their Merkle commitments and linear combinations within sub-prover communication sublinear in N. Encouragingly, the following encoding pattern fits our requirement and could be a game changer.

• **Ligero [1] and logarithmic randomness [18].** 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 C with block length n, one can collect n' codewords of C as a $\mathbb{F}^{n' \times n}$-matrix $\hat{\mathbf{U}}$, and view it as a codeword of the interleaved linear code $C^{n'}$, where each row $\hat{\mathbf{U}}[i, \cdot]$ is a codeword in C. To test the membership of $\hat{\mathbf{U}}$ in $C^{n'}$, 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 column $\hat{\mathbf{U}}_j$, verifies if $\langle \mathbf{r}, \hat{\mathbf{U}}_j \rangle = \hat{\mathbf{u}}[j]$. Following this pattern, Diamond et al. [18] replace the linear randomness $\mathbf{r}$ with one generated by *logarithmic randomness*, i.e., samples $\mathbf{r}' \leftarrow_{\$} \mathbb{F}^{\log n'}$ and defines 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 allocate 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 here is $M \le n'$. With this customized distribution method for interleaved code-based IOPP, it takes only $O(\lambda \cdot n')$ 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 [11, 23] 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 strategies that shrink proof length without sacrificing the previous IOPP. Fortunately, the following work presents a possible solution.

• **Ligerito [32].** Very 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 $f \in \mathcal{F}_{\mu}^{(\le 1)} := \mathbb{F}^{(\le 1)}[x_1, \cdots, x_\mu]$, a value $y \in \mathbb{F}$ and a point $\mathbf{z} \in \mathbb{F}^{\mu}$, it holds that $y = f(\mathbf{z}) = \sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) \cdot z(\mathbf{b}) = \langle \mathbf{u}, \mathbf{z}' \rangle$, where $B_\mu := \{0,1\}^\mu$, $z \in \mathcal{F}_{\mu}^{(\le 1)}$ is the Lagrange interpolation of vector $\mathbf{z}' := \otimes_{i=1}^{\mu}(1 - \mathbf{z}[i], \mathbf{z}[i]) \in \mathbb{F}^{2^\mu}$, and $\mathbf{u} = \mathbf{f} := f(B_\mu) = (f(00\cdots00), f(00\cdots01) \cdots, f(11\cdots11)) \in \mathbb{F}^{2^\mu}$ are sequential evaluations of f over B_μ. The protocol begins by mandating that $\mathcal{P}$ "partially" run with $\mathcal{V}$ a standard SumCheck protocol for $\mu' < \mu$ rounds, reducing the claim $y = \langle \mathbf{u}, \mathbf{z}' \rangle$ to

$$
\begin{aligned}
y' &= \sum_{\mathbf{b} \in B_{\mu - \mu'}} f(\mathbf{r}, \mathbf{b}) \cdot z(\mathbf{r}, \mathbf{b}) \\
&= \langle \mathbf{r}_{\otimes} \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu - \mu'}}(\mathbf{u}), \mathbf{r}_{\otimes} \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu - \mu'}}(\mathbf{z}') \rangle := \langle \mathbf{u}', \mathbf{w} \rangle,
\end{aligned}
$$

where $\mathbf{r} \in \mathbb{F}^{\mu'}$ is the randomness generated in μ' rounds of SumCheck, and $\mathbf{r}_{\otimes} := \otimes_{i=1}^{\mu'}(1 - \mathbf{r}[i], \mathbf{r}[i]) \in \mathbb{F}^{2^{\mu'}}$. To check this claim, it suffices for $\mathcal{V}$ to acquire from $\mathcal{P}$ the vector $\mathbf{u}'$, because $\mathbf{r}_{\otimes}$ and $\mathbf{z}$ are known to $\mathcal{V}$, hence so does $\mathbf{w}$. 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 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 $y' = \langle \mathbf{u}', \mathbf{w} \rangle$.

Authors of Ligerito notice the following room for iteration: the verification for $y' = \langle \mathbf{u}', \mathbf{w} \rangle$ and $\mathbf{r}_\otimes \cdot \hat{\mathbf{U}}_I = \mathbf{u}' \cdot \hat{\mathbf{G}}_I$ are essentially $|I| + 1$ inner products w.r.t. $\mathbf{u}'$, which now aligns with the evaluations of the polynomial $f(\mathbf{r}, \cdot) \in \mathcal{F}^{(\leq 1)}_{\mu-\mu'}$ over $B_{\mu-\mu'}$. Therefore, the verification can be reduced to a new inner product

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

where $\mathbf{s} \in \mathbb{F}^{|I|+1}$ is a random vector for batching. The resulting inner product shares a similar form with $y = \langle \mathbf{u}, \mathbf{z}' \rangle$, yet with a $2^{-\mu'}$-fold scale. With this observation, one can iteratively reduce the inner product interleaved with an off-and-on-ed SumCheck for f, until its scale is tiny enough for a direct verification.

Challenges ahead. The distribution for Ligerito, which involves that for partial SumCheck and vector-matrix multiplication within Ligero IOPP, is nontrivial. Strategies for distributing standard SumCheck are anything but new. As first introduced in [42] and later applied in [27, 40, 46], the central point is to number $M = 2^m$ sub-provers with m-bit binary expression of their indices, i.e., $\mathbf{bin}_m(i) := (b_{m-1}, \cdots, b_0)$ for $\mathcal{P}_i$, where $i = \sum_{j=0}^{m-1} 2^j b_j$. After that, it holds

$$\sum_{\mathbf{b} \in B_\mu} f(\mathbf{b}) = \sum_{i=0}^{M-1} \sum_{\mathbf{b}' \in B_{\mu-m}} f(\mathbf{b}', \mathbf{bin}_m(i)) =: \sum_{i=0}^{M-1} \sum_{\mathbf{b}' \in B_{\mu-m}} f^{(i)}(\mathbf{b}'),$$

so each $\mathcal{P}_i$ can process most computation locally w.r.t. $f^{(i)}$, and the master sub-prover $\mathcal{P}_0$ is delegated to complete the final $m = \log M$ rounds of interaction with $\mathcal{V}$. However, its direct application to Ligerito heavily burdens $\mathcal{P}_0$, since partial SumCheck is essentially a standard one with an on-and-off-ed pattern, which produces multiple such "final m rounds". Additionally, careful arrangement for distributing Ligero IOPP to achieve M-independent proof size during commitment and querying phases is required; as for communication, although $O(\lambda \cdot 2^{\mu'} + M \cdot 2^{\mu-\mu'})$ is already sublinear in $N = 2^\mu$, there remains room to reduce constant overheads, by avoiding the master $\mathcal{P}_0$'s abundant information gathering and broadcasting.

Aside from distribution, the $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ proof size comes at the cost of a $O(N)$ verification time. It essentially results from a direct computation for $\mathbf{w} = \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{z}')$, which costs $O(2^\mu)$ field operations. Therefore, verification requires further simplification to achieve succinctness.

Our construction. Now, we overview our ideas on addressing the aforementioned challenges, and obtain a distribution framework for code-based multilinear PCS with communication sublinear in N and proof size independent of M.

• **Distributed partial SumCheck.** In each iteration, a μ'-round partial SumCheck reduces 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 [42] 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 being 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 beginning $\mu-k-m$ levels of summation can be distributed among sub-provers, until they get stuck, with each holding $s_k^{(i)}(x_k) = \sum_{\mathbf{b}' \in B_{\mu-k-m}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b}', \mathbf{bin}_m(i))$. To proceed, we *buffer* the rest m level 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 $\mathcal{P}_i$, and $\mathcal{P}_i$ 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 Sect. 3.1.

• **Distributed vector-matrix multiplication.** Parallel to partial SumCheck, in each iteration $\{\mathcal{P}_i\}_{i=0}^{M-1}$ also convince $\mathcal{V}$ that $\mathbf{u} = \mathbf{r}_\otimes \cdot \mathbf{U}$ where $\mathbf{U} \in \mathbb{F}^{2^{\mu'} \times 2^{\mu-\mu'}}$, by invoking an IOPP for the row-encoded matrix $\hat{\mathbf{U}}$. As sketched before, we use the proof aggregation method specialized from interleaved code, by assigning each $\mathcal{P}_i$ $2^{\mu'-m}$ rows, 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 columns of $\hat{\mathbf{U}}$ column-wise and obtain a vector of hash values $\mathbf{h}$, and builds a Merkle tree with $\mathbf{h}$ being leaves. For distribution, we let each $\mathcal{P}_i$ process $\hat{\mathbf{U}}^{(i)} = \mathsf{Enc}(\mathbf{U}^{(i)})$ and obtain $\mathbf{h}^{(i)}$. At this point, directly computing M Merkle trees results in an M-fold proof size. To avoid this overhead, we equidistantly partition columns. That is, after each $\mathcal{P}_i$ obtains $\mathbf{h}^{(i)}$, they send the j-th slice of $\mathbf{h}^{(i)}$ to another sub-prover $\mathcal{P}_j$, so each $\mathcal{P}_j$ holds several "semi-hashed" columns of $\hat{\mathbf{U}}$, and proceeds to complete column hashing.

Linear combination follows a similar pattern, 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 another matrix $\mathbf{U}' \in \mathbb{F}^{2^{\mu''} \times 2^{\mu-\mu'-\mu''}}$ 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 adding $\{\mathbf{u}^{(i)}\}_i$ and broadcasting the result to other sub-provers.

We provide detailed explanation for these three components of distributed vector-matrix multiplication in Sect. 3.2, and combine them with distributed partial SumCheck to obtain Veloz in Sect. 3.3.

• **Achieving an efficient verifier.** While distribution benefits prover side, verification is linear in N, mainly due to computing the vector $\mathbf{w} = \mathbf{r}_\otimes \cdot \mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{z}')$. Towards this, authors of [32] point out the following tensor structure of $\mathbf{z}'$: $\mathbf{Mat}_{2^{\mu'} \times 2^{\mu-\mu'}}(\mathbf{z}') = \otimes_{i=1}^{\mu'} (1 - \mathbf{z}[i], \mathbf{z}[i])^\top \cdot \otimes_{i=\mu'+1}^{\mu} (1 - \mathbf{z}[i], \mathbf{z}[i])$. 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, $\mathbf{w}$ can be efficiently computed. Nonetheless, verification still costs $O(\lambda \cdot \frac{N \log \log N}{\log^2 N})$, if we desire a $O(\lambda \cdot \frac{\log^2 N}{\log \log N})$ proof size. It does not leave much room for further optimization, because in the next iteration, $\mathbf{z}''$ is a random linear combination of 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 hurriedly 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 the matrix $\mathbf{Mat}_{2^{\mu''} \times 2^{\mu-\mu'-\mu''}}(\mathbf{z}'')$, we can exchange 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 Sect. 3.4 and Sect. 4.

1.3 Related Works

Pairing-based PCSs. Kate et al. [25] first propose the concept of PCS, and present one construction for univariate polynomials based on bilinear pairing. Papamanthou et al. [33] 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 [26] 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) [6] pioneers the former category, followed by subsequent DEEP-FRI [10], STIR [3] and WHIR [4], that improve soundness analysis for better verification efficiency and smaller proof size. BaseFold [47] generalizes RS code-based IOPPs to *foldable* code-based ones, and features field agnosticity. Compared with FRI, BaseFold verifier makes more queries to lower soundness error. Its soundness is improved in DeepFold [24], 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 [1] first studies interleaved code-based IOPPs. The idea is then refined by Bootle et al. [11] and in Brakedown [23]. 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 [13] improves Brakedown on binary fields using the code-switching technique [35], which obtains polylogarithmic proof size.

Distributed PCSs. DIZK [41] first explores the distributed PCS and its application in distributed SNARK. Subsequent works include Pianist [30], Hekaton [36], HyperPianist [27], Cirrus [40] and Soloist [29]. All constructions are built on [25] and [33] for verification efficiency and proof aggregation, benefiting from homomorphisms in bilinear groups. As an exception, HyperPianist also distributes [26], and is the only one among above with a transparent setup.

Another line of works distributes code-based PCSs. deVirgo [42] first considers their distribution, and proposes a solution for FRI [6]-based multilinear PCS. Recently, Li et al. introduce $\text{DePIP}_{\text{FRI}}$ [28], 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 [28], FRIttata [45] distributes bivariate polynomials built on FRI, and HyperFond [46] distributes BaseFold [47] 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.

2 Preliminaries

Notations. Let λ denote the security parameter. A function $f(\lambda) = \text{poly}(\lambda)$ if there exists some $c \in \mathbb{N}$ such that $f(\lambda) = O(\lambda^c)$. If for all $c \in \mathbb{N}, f(\lambda) = o(\lambda^{-c})$, then $f(\lambda) = \text{negl}(\lambda)$ is called *negligible*. The probability of $1 - \text{negl}(\lambda)$ is *overwhelming*. A *relation* $\mathcal{R}$ is a set consisting of instance-witness pairs $(\mathbb{x}; \mathbb{w})$.

Let $\mathbb{F}$ be a finite field such that $\log|\mathbb{F}| = \Omega(\lambda)$. Denote by $\mathcal{F}_\mu^{(\le d)}$ the set $\mathbb{F}^{(\le d)}[x_1, \cdots, x_\mu]$ of μ-variate polynomials with $\deg(x_i) \le d$ for each i. Let $r \leftarrow_\$ S$ denote that r is sampled uniformly at random from a finite set S. For $i, j \in \mathbb{N}$ and $i < j$, $[i : j]$ denotes the discrete set $\{i, i+1, \cdots, j-1\}$. We use lowercase boldface letters to denote vectors $\mathbf{u} = (\mathbf{u}[1], \cdots, \mathbf{u}[\mu]) \in \mathbb{F}^\mu$, and uppercase boldface letters $\mathbf{V}$ to denote matrices. For a vector $\mathbf{u} \in \mathbb{F}^{\mu \cdot \nu}$, let $\mathbf{Mat}_{\mu\times\nu}(\mathbf{u}) \in \mathbb{F}^{\mu\times\nu}$ denote the row-major reshape of $\mathbf{u}$. For a matrix $\mathbf{U}$, $\mathbf{U}[i : j, \cdot]$ denotes its submatrix consisting of the i-th to $(j-1)$-th rows. Given an index set $I = \{i_1, \cdots, i_{|I|}\}$, let $\mathbf{U}_I$ be a matrix concatenated by i_1-th, $\cdots$, $i_{|I|}$-th columns of $\mathbf{U}$.

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 multilinear polynomial $\tilde{f} \in \mathcal{F}_\mu^{(\le 1)}$ 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.

Lemma 1 (Schwartz-Zippel Lemma). *Let $f \in \mathcal{F}_\mu^{(\le d)}$ be a non-zero polynomial and $S \subseteq \mathbb{F}$ a finite subset of $\mathbb{F}$, then*

$$\Pr[f(\mathbf{r}[1], \cdots, \mathbf{r}[\mu]) = 0] \le \frac{d\mu}{|S|},$$

where $\mathbf{r}[1], \cdots, \mathbf{r}[\mu]$ are chosen uniformly at random from S.

Multivariate SumCheck protocol [31]. 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 $f \in \mathcal{F}_\mu^{(\le d)}$. In more detail, $\mathcal{P}$ and $\mathcal{V}$ engage in a μ-round reduction as follows.

In round k from 1 to $\mu - 1$, $\mathcal{P}$ computes a univariate polynomial $s_k(x_k) := \sum_{\mathbf{b}\in B_{\mu-k}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b})$, and sends it to $\mathcal{V}$. Then $\mathcal{V}$ verifies whether $\deg(s_k) \le d$ and $s_k(0) + s_k(1) = s_{k-1}(\mathbf{r}[k-1])$, where $s_1(0) + s_1(1) = v$ for $k = 1$. If the 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 the 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 [18, 47]. 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 $C \subseteq \mathbb{F}^n$. C is associated with a generator matrix $\mathbf{G} \in \mathbb{F}^{k\times n}$, of which the rows form a basis of C, and the encoding of $\mathbf{m} \in \mathbb{F}^k$ is computed as $\mathbf{m} \cdot \mathbf{G}$. The *code rate* of C is defined as $\rho := \frac{k}{n} \le 1$, and the *minimum Hamming distance* of C is the minimum number of different entries between any $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$. If C has a minimum Hamming distance $d \in [0 : n+1]$, we call C a $[n, k, d]$-*code*, and denote by $\delta_C := \frac{d}{n}$ the *relative minimum distance* of C. Specifically, if $d = n - k + 1$, then C is *maximum distance separable* (MDS).

Given a $[n, k, d]$-code C and an integer $m \ge 1$, the corresponding *m-fold interleaved code* is defined as $C^m \subseteq (\mathbb{F}^n)^m$. Viewing the elements of 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 relative distance $\frac{\delta}{n}$ from C^m, denoted by $\Delta^m_C(\mathbf{M}, C^m) \le \frac{\delta}{n}$, if $\exists \mathbf{M}^* \in C^m$ such that $\mathbf{M}$ and $\mathbf{M}^*$ differ at up to δ columns.

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 equidistantly partition $\mathbf{w}$ into μ slices $(\mathbf{w}_1 \| \cdots \| \mathbf{w}_\mu) \in (\mathbb{F}^\nu)^\mu$, so $\langle \mathbf{u} \otimes \mathbf{v}, \mathbf{w}\rangle = \sum_{i=1}^{\mu} \mathbf{u}[i] \cdot \langle \mathbf{v}, \mathbf{w}_i\rangle = \sum_{i=1}^{\mu} \langle \mathbf{u}[i] \cdot \mathbf{w}_i, \mathbf{v}\rangle = \mathbf{u} \cdot \mathbf{Mat}_{\mu\times\nu}(\mathbf{w}) \cdot \mathbf{v}^\top$, which only costs $\mu\nu + \min(\mu, \nu)$ multiplications.

Another observation is that, given $\{\mathbf{u}_i, \mathbf{v}_i \in \mathbb{F}^{\mu_i}\}_{i=1}^k$, we have

$$\begin{aligned}\langle \otimes_{i=1}^k \mathbf{u}_i, \otimes_{i=1}^k \mathbf{v}_i\rangle =& \langle \mathbf{u}_1[1] \cdot \otimes_{i=2}^k \mathbf{u}_i \| \cdots \| \mathbf{u}_1[\mu_1] \cdot \otimes_{i=2}^k \mathbf{u}_i,\\ & \mathbf{v}_1[1] \cdot \otimes_{i=2}^k \mathbf{v}_i \| \cdots \| \mathbf{v}_1[\mu_1] \cdot \otimes_{i=2}^k \mathbf{v}_i\rangle\\ =& \langle \mathbf{u}_1, \mathbf{v}_1\rangle \cdot \langle \otimes_{i=2}^k \mathbf{u}_i, \otimes_{i=2}^k \mathbf{v}_i\rangle = \prod_{i=1}^k \langle \mathbf{u}_i, \mathbf{v}_i\rangle,\end{aligned}$$

so the number of multiplications is only $\sum_{i=1}^k \mu_i + k - 1$, compared to $3 \cdot \prod_{i=1}^k \mu_i$ by a direct computation.

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

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

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 $f \in \mathcal{F}_\mu^{(\le 1)}$, 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 [18, 47]). *A multilinear polynomial commitment scheme* PCS *is a tuple of polynomial time algorithms* (Setup, Commit, Open, Eval) *defined as:*

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

$(\mathsf{cm}, t) \leftarrow \mathsf{Commit}(\mathsf{gp}, f)$*: on input public parameters* gp *and a multilinear polynomial* f*, it 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*, it outputs a bit* b *denoting 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 its opening hint* t*, and tries to convince* $\mathcal{V}$ *that* cm *is a commitment to* f *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* $f \in \mathcal{F}_\mu^{(\le 1)}$ *and* $\mathbf{z} \in \mathbb{F}^\mu$*, it holds*

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

PCS *is binding if for any* $\mu \in \mathbb{N}$ *and PPT adversary* $\mathcal{A}$*, it holds*

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

PCS *is knowledge sound if* Eval *is an argument of knowledge for the relation* $\mathcal{R}_{\mathsf{Eval},\mathsf{gp}}$ *defined as*

$$\left\{\left(\mathbb{x} = (\mathsf{cm}, \mathbf{z}, y); \mathbb{w} = (f, t)\right) : f(\mathbf{z}) = y \wedge \mathsf{Open}(\mathsf{gp}, \mathsf{cm}; f, t) = 1\right\}.$$

For other related definitions as argument of knowledge and interactive oracle proof of proximity, please refer to Appendix A.

3 Veloz: Our Distribution Framework for Code-Based Multilinear PCS

In this section, we introduce Veloz, a distribution framework for code-based multilinear PCS, and its two instantiations, $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$. As in the overview of our construction in Sect. 1.2, we first distribute a partial Sumcheck reducing μ' variables from a μ-variate polynomial in Sect. 3.1. Then, we show a distributed protocol proving the correctness of vector-matrix multiplication in Sect. 3.2. In Sect. 3.3, we combine these components and present a distribution framework, Veloz, for code-based multilinear PCS, which can be instantiated with any efficient linear code. Finally in Sect. 3.4, we introduce $\mathsf{Veloz}_{\mathsf{RS}}$ and $\mathsf{Veloz}_{\mathsf{Lin}}$, and in particular design an efficient verification algorithm for $\mathsf{Veloz}_{\mathsf{RS}}$. In the following, we denote the number of sub-provers by $M = 2^m$, and number each sub-prover $\mathcal{P}_i$ with $\mathbf{bin}_m(i) := (b_{m-1}, \cdots, b_0)$, where $i = \sum_{j=0}^{m-1} 2^j b_j$.

3.1 Distributed Partial SumCheck

Given two multilinear polynomials $f, z \in \mathcal{F}_\mu^{(\le 1)}$, a μ'-round partial SumCheck between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ reduces $y = \sum_{\mathbf{b}\in B_\mu} f(\mathbf{b}) \cdot z(\mathbf{b})$ 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 Sect. 2 that in the k-th round of SumCheck where $1 \le k \le \mu'$, 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 $s_k(x_k) =$

Protocol 1: Distributed partial SumCheck. Given two multilinear polynomials $f, z \in \mathcal{F}_\mu^{(\leq 1)}$, we denote $g := f \cdot z \in \mathcal{F}_\mu^{(\leq 2)}$. 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. Initially, each $\mathcal{P}_i$ determines $f^{(i)}(\cdot) := \sum_{\mathbf{b} \in B_{\mu-\mu'-m}} f(\cdot, \mathbf{b}, \mathbf{bin}_m(i)) \in \mathcal{F}_{\mu'}^{(\leq 1)}$. Similarly for $z^{(i)}(\cdot)$.
2. In round $1 \leq k \leq \mu'$, $s_k(x_k)$ is computed and verified as follows
- For all $\mathbf{b} \in B_{\mu'-k}$ and $i \in [0 : M]$, $\mathcal{P}_i$ computes $s_{k,f,\mathbf{b}}^{(i)}(x_k) = (1 - x_k) \cdot f^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], 0, \mathbf{b}) + x_k \cdot f^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], 1, \mathbf{b})$. Similarly for $s_{k,z,\mathbf{b}}^{(i)}(x_k)$. Then, each $\mathcal{P}_i$ computes $s_k^{(i)}(x_k) = \sum_{\mathbf{b} \in B_{\mu'-k}} s_{k,f,\mathbf{b}}^{(i)}(x_k) \cdot s_{k,z,\mathbf{b}}^{(i)}(x_k)$.
- For $j = 1, \cdots, m$, for all $i < 2^{m-j}$, $\mathcal{P}_i$ acquires $s_k^{(i+2^{m-j})}(x_k)$ from $\mathcal{P}_{i+2^{m-j}}$, and updates $s_k^{(i)}(x_k) \leftarrow s_k^{(i)}(x_k) + s_k^{(i+2^{m-j})}(x_k)$. After that, $\mathcal{P}_0$ sends $s_k(x_k) = s_k^{(0)}(x_k)$ to $\mathcal{V}$, and $\mathcal{V}$ verifies $s_k(0) + s_k(1) = s_{k-1}(\mathbf{r}[k-1])$ ($s_1(0) + s_1(1) = y$ when $k = 1$). If verification passes, $\mathcal{V}$ responds with $\mathbf{r}[k] \leftarrow_\$ \mathbb{F}$, and each $\mathcal{P}_i$ updates $f^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b}) \leftarrow f^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], \mathbf{r}[k], \mathbf{b})$ for all $\mathbf{b} \in B_{\mu'-m}$. Similarly for $z^{(i)}(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b})$.
3. $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ set $y' := s_{\mu'}(\mathbf{r}[\mu'])$ and $\mathbf{r} := (\mathbf{r}[1], \cdots, \mathbf{r}[\mu'])$.

Figure 1: Distributed partial SumCheck

$\sum_{\mathbf{b} \in B_{\mu-k}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b})$ for $\mathcal{V}$, and receive the randomness $\mathbf{r}[k]$. Previous distribution strategies [27, 40, 42, 46] mandate that each sub-prover $\mathcal{P}_i$ compute $s_k^{(i)}(x_k) = \sum_{\mathbf{b} \in B_{\mu-k-m}} f(\mathbf{r}[1], \cdots, \mathbf{r}[k-1], x_k, \mathbf{b}, \mathbf{bin}_m(i))$ 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 can 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 $s_k^{(i)}(x_k) \leftarrow s_k^{(i)}(x_k) + s_k^{(i+\frac{M}{4})}(x_k)$;

$\vdots$

m. $\mathcal{P}_0$ acquires $s_k^{(1)}(x_k)$ from $\mathcal{P}_1$, and 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. Note that the buffered computation only adjusts the destinations where polynomials are sent, hence requiring *no* extra communication cost. Following this strategy, the detailed distribution for partial SumCheck is given in Protocol 1 in Fig. 1.

Protocol 2: Distributed commitment and querying.

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

Querying phase. On receiving $\mathcal{V}$'s indexed set $I \subseteq [0 : \frac{2^{\mu-\mu'}}{\rho}]$, $\mathcal{P}_0$ first collects the columns $(\hat{\mathbf{U}}^{(i)})_I$ from each $\mathcal{P}_i$, and concatenates them to obtain $(\hat{\mathbf{U}})_I$. Then, $\mathcal{P}_0$ sends $(\hat{\mathbf{U}})_I$ and $|I|$ Merkle paths to $\mathcal{V}$. For each index $p \in I$, $\mathcal{V}$ computes $\mathbf{h}[p]$ using the column $\hat{\mathbf{U}}_p$, and verifies its Merkle path to $\mathsf{MC}(\hat{\mathbf{U}})$.

Figure 2: Distributed commitment and querying

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 $\mathbf{u} = \mathbf{r}_\otimes \cdot \mathbf{U} \in \mathbb{F}^{2^{\mu-\mu'}}$, 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 C, each sub-prover $\mathcal{P}_i$ encodes $2^{\mu'-m}$ rows of $\mathbf{U}$, i.e., computing $\hat{\mathbf{U}}^{(i)} := \mathbf{U}[i \cdot 2^{\mu'-m} : (i+1) \cdot 2^{\mu'-m}, \cdot] \cdot \mathbf{G}$, 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. In the single prover setting, to prove that each row of $\hat{\mathbf{U}} = \mathbf{U} \cdot \mathbf{G} \in \mathbb{F}^{2^{\mu'} \times \frac{2^{\mu-\mu'}}{\rho}}$ is close to the code space 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 [23] proceeds by first hashing each column of $\hat{\mathbf{U}}$ 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, the same technique does not directly apply to the distributed setting: after each $\mathcal{P}_i$ hashes the columns of $\hat{\mathbf{U}}^{(i)} = \hat{\mathbf{U}}[i \cdot 2^{\mu'-m} : (i+1) \cdot 2^{\mu'-m}, \cdot]$ 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 the roots $\{\mathsf{MC}(\mathbf{h}^{(i)})\}_{i=0}^{M-1}$ to $\mathcal{V}$. The resulting proof size amounts to $O(\lambda \cdot$

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 $\mathbf{u} = \mathbf{r}_\otimes \cdot \mathbf{U} \in \mathbb{F}^{2^{\mu-\mu'}}$ as follows.

1. Each $\mathcal{P}_i$ computes $\mathbf{r}_\otimes^{(i)} = \tilde{eq}_{\mathbf{bin}_m(i)}(r[1], \cdots, r[m]) \cdot \otimes_{j=m+1}^{\mu'}(1 - r[j], r[j]) \in \mathbb{F}^{2^{\mu'-m}}$, and $\mathbf{u}^{(i)} = \mathbf{r}_\otimes^{(i)} \cdot \mathbf{U}^{(i)} \in \mathbb{F}^{2^{\mu-\mu'}}$. For all $j \neq i$, $\mathcal{P}_i$ sends $\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$ to $\mathcal{P}_j$.
2. For each $j \in [0 : M]$, $\mathcal{P}_j$ sums $\{\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]\}_{i=0}^{M-1}$ up for $\mathbf{u}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$.

Figure 3: Distributed linear combination

$(2^{\mu'} + M \cdot \log \frac{2^{\mu-\mu'}}{\rho}))$. The principal part of proof size, $2^{\mu'}$, 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 in distribution.

We observe that M exists essentially because the columns are not completely hashed: values in $\mathbf{h}^{(i)}$ are directly used as Merkle leaves. To address this, we force a complete hashing by mandating that $\mathcal{P}_i$ sends $\mathcal{P}_j$ $\mathbf{h}^{(i)}[j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$, the j-th slice of $\mathbf{h}^{(i)}$. Each $\mathcal{P}_j$ then computes $\mathbf{h}[k] := \mathbb{H}_M(\mathbf{h}^{(0)}[k], \cdots, \mathbf{h}^{(M-1)}[k])$ for all $k \in [j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$. Now, with $\mathbf{h} = (\mathbf{h}[0], \cdots, \mathbf{h}[\frac{2^{\mu-\mu'}}{\rho} - 1])$ equidistantly scattered in $\{\mathcal{P}_i\}_{i=0}^{M-1}$, each $\mathcal{P}_j$ continues building a Merkle tree with $\mathbf{h}[j \cdot \frac{2^{\mu-\mu'-m}}{\rho} : (j+1) \cdot \frac{2^{\mu-\mu'-m}}{\rho}]$. Finally, $\mathcal{P}_0$ gathers M roots, views them as leaves, and computes $\mathsf{MC}(\hat{\mathbf{U}})$ for $\mathcal{V}$. The derived proof size of querying $\Theta(\lambda)$ columns now reduces back 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 $\mathcal{P}$ needs to linearly combine the rows of $\mathbf{U}$ with the 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, we let each $\mathcal{P}_i$ compute $\mathbf{u}^{(i)} = \mathbf{r}_\otimes^{(i)} \cdot \mathbf{U}^{(i)} \in \mathbb{F}^{2^{\mu-\mu'}}$. To obtain $\mathbf{u}$, it suffices to let $\mathcal{P}_0$ gather $\{\mathbf{u}^{(i)}\}_{i=0}^{M-1}$ for summation. However, the derived $\mathbf{u}$ will be reshaped into another matrix $\mathbf{U}' = \mathbf{Mat}_{2^{\mu''} \times 2^{\mu-\mu'-\mu''}}(\mathbf{u})$ 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 the encoded matrix $\hat{\mathbf{U}}$ is hashed into one value. Therefore, we can mimic the aforementioned element exchanging process as follows. Let each $\mathcal{P}_i$ send $\mathcal{P}_j$ the j-th slice of $\mathbf{u}^{(i)}$, $\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$. Then, each $\mathcal{P}_j$ sums $\{\mathbf{u}^{(i)}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]\}_{i=0}^{M-1}$ to obtain $\mathbf{u}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$.

After rearrangement, communication halves from $2M \cdot 2^{\mu-\mu'}$ to $M \cdot 2^{\mu-\mu'}$. Coming with this reduction is another advantage: we have obtained $\mathbf{U}'$ *for free*, because $\mathbf{u}[j \cdot 2^{\mu-\mu'-m} : (j+1) \cdot 2^{\mu-\mu'-m}]$ 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 above to present our distribution framework, Veloz, for multilinear PCS. In each round of iteration below, we use Enc_k to denote the encoding algorithm instantiated with an efficient linear code C_k, including RS code and Brakedown code [23].

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

$(\mathsf{cm}; t) \leftarrow \mathsf{Commit}(\mathsf{gp}, f)$: on input the public parameters gp and a multilinear polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$, it parses f's Lagrange coefficients as $\mathbf{f} = f(B_{\mu_0}) = \mathbf{u}_0 \in \mathbb{F}^{2^{\mu_0}}$, reshapes $\mathbf{U}_1 = \mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{u}_0)$, computes $\hat{\mathbf{U}}_1 = \mathsf{Enc}_1(\mathbf{U}_1)$, and outputs a commitment $\mathsf{cm} = \mathsf{MC}(\hat{\mathbf{U}}_1)$ and an 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 an opening hint t, it reshapes the Lagrange coefficients $\mathbf{f} \in \mathbb{F}^{2^{\mu_0}}$ as a matrix $\mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{f})$, and outputs $b = 1$ if both (1) $\mathsf{cm} = \mathsf{MC}(t)$ and (2) $\Delta_{C_1}^{\mu'_1}(\mathsf{Enc}_1(\mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{f})), t) < \frac{\delta_{C_1}}{2}$ hold (recall that $\Delta_{C_1}^{\mu'_1}(\cdot, \cdot)$ denotes measuring the relative distance of two matrices with the μ'_1-fold interleaved code $C_1^{\mu'_1}$); 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 $\{\mathcal{P}\}_{i=0}^{M-1}$ 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 $\{\mathcal{P}\}_{i=0}^{M-1}$ 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 cm is a commitment to f and that $y_1 = f(\mathbf{z})$.

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 Sect. 3.1 and the distributed vector-matrix multiplication in Sect. 3.2. We formally present the protocol in Fig. 4, where the iteratively reduced claims are highlighted in blue.

3.4 Instantiations with RS and Brakedown codes

We now instantiate Veloz with RS code and the fast linear code from Brakedown [23], and obtain Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$, respectively. The metrics including proving time, communication and proof size can be analyzed independent from code choices, so we summarize them in Sect. 4.2. What we should focus on, however, is the different verification efficiency of Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$, as we now explain.

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 $f \in \mathcal{F}_{\mu_0}^{(\le 1)}$ and $t = \hat{\mathbf{U}}_1 = \mathsf{Enc}_1(\mathbf{U}_1)$.
With $\mathbf{u}_0 = \mathbf{f} \in \mathbb{F}^{2^{\mu_0}}$, $\mathbf{U}_1 = \mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{u}_0)$, and $\mathbf{z}_1 = \mathbf{z}_\otimes = \otimes_{i=1}^{\mu_0}(1 - \mathbf{z}[i], \mathbf{z}[i])$, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ convince $\mathcal{V}$ that $y_1 = f(\mathbf{z}) = \langle \mathbf{u}_0, \mathbf{z}_1 \rangle$ as follows.
1. In round $1 \le k \le \ell - 2$, the claim $y_k = \langle \mathbf{u}_{k-1}, \mathbf{z}_k \rangle$ is reduced to $y_{k+1} = \langle \mathbf{u}_k, \mathbf{z}_{k+1} \rangle$:

- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ distributedly encode $\mathbf{U}_k \in \mathbb{F}^{2^{\mu'_k} \times 2^{\mu_k}}$ to get $\hat{\mathbf{U}}_k = \mathsf{Enc}_k(\mathbf{U}_k)$, and invoke the commitment phase from Protocol 2 to obtain and send $\mathsf{MC}(\hat{\mathbf{U}}_k)$ to $\mathcal{V}$. (This step can be omitted when $k = 1$ because $\mathsf{cm} = \mathsf{MC}(\hat{\mathbf{U}}_1)$.)
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ run a μ'_k-round distributed partial SumCheck for $f(\mathbf{r}_1 \| \cdots \| \mathbf{r}_{k-1}, \cdot)$ and $z_k(\cdot) \in \mathcal{F}_{\mu_{k-1}}^{(\le 1)}$ 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 $y'_k \in \mathbb{F}$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ 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 $\mathbf{s}_k := (1, \gamma_k, \cdots, \gamma_k^{|I_k|}) \in \mathbb{F}^{|I_k|+1}$.
- With $\mathbf{r}_k \in \mathbb{F}^{\mu'_k}$ and $\mathbf{s}_k \in \mathbb{F}^{|I_k|+1}$, 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 $\mathbf{z}_{k+1} = ((\mathbf{G}_k)_{I_k} \| \mathbf{w}_k^\top) \cdot \mathbf{s}_k \in \mathbb{F}^{2^{\mu_k}}$. Separately, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ invoke Protocol 3 to distributedly compute $\mathbf{u}_k = (\mathbf{r}_k)_\otimes \cdot \mathbf{U}_k \in \mathbb{F}^{2^{\mu_k}}$, and reshape it into $\mathbf{U}_{k+1} = \mathbf{Mat}_{2^{\mu'_{k+1}} \times 2^{\mu_{k+1}}}(\mathbf{u}_k)$; $\mathcal{V}$ 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 $\mathbf{U}_{\ell-1} = \mathbf{Mat}_{2^{\mu'_{\ell-1}} \times 2^{\mu_{\ell-1}}}(\mathbf{u}_{\ell-2})$.

- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ distributedly encode $\mathbf{U}_{\ell-1} \in \mathbb{F}^{2^{\mu'_k} \times 2^{\mu_k}}$ to get $\hat{\mathbf{U}}_{\ell-1} = \mathsf{Enc}_{\ell-1}(\mathbf{U}_{\ell-1})$, and invoke the commitment phase from Protocol 2 to obtain and send $\mathsf{MC}(\hat{\mathbf{U}}_\ell)$ to $\mathcal{V}$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ and $\mathcal{V}$ run a $\mu'_{\ell-1}$-round distributed partial SumCheck for $f(\mathbf{r}_1 \| \cdots \| \mathbf{r}_{\ell-2}, \cdot)$ and $z_{\ell-1}(\cdot) \in \mathcal{F}_{\mu_{\ell-2}}^{(\le 1)}$ as in Protocol 1. After that, both sides obtain randomness $\mathbf{r}_{\ell-1} \in \mathbb{F}^{\mu'_{\ell-1}}$ and the reduced value $y'_{\ell-1} \in \mathbb{F}$.
- $\{\mathcal{P}_i\}_{i=0}^{M-1}$ 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, both sides compute $\mathbf{w}_{\ell-1} = (\mathbf{r}_{\ell-1})_\otimes \cdot \mathbf{Mat}_{2^{\mu'_{\ell-1}} \times 2^{\mu_{\ell-1}}}(\mathbf{z}_{\ell-1})$. $\{\mathcal{P}_i\}_{i=0}^{M-1}$ additionally compute the linear combination $\mathbf{u}_{\ell-1} = (\mathbf{r}_{\ell-1})_\otimes \cdot \mathbf{U}_{\ell-1} \in \mathbb{F}^{2^{\mu_{\ell-1}}}$ using Protocol 3. 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) $y'_{\ell-1} = \langle \mathbf{w}_{\ell-1}, \mathbf{u}_{\ell-1} \rangle$.

3. In round $k = \ell$, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ directly send $\mathbf{u}_{\ell-1} \in \mathbb{F}^{2^{\mu_{\ell-1}}}$ to $\mathcal{V}$, and $\mathcal{V}$ verifies (1) $\wedge$ (2) above.

Figure 4: Distributed evaluation protocol

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 [32], a possible solution is to exploit the following structure: given $\mathbf{z}_1 = \mathbf{z}_\otimes = \otimes_{i=1}^{\mu_0}(1 - \mathbf{z}[i], \mathbf{z}[i])$, it holds that

$$\mathbf{Mat}_{2^{\mu'_1} \times 2^{\mu_1}}(\mathbf{z}_1) = \otimes_{i=1}^{\mu'_1}(1 - \mathbf{z}[i], \mathbf{z}[i])^\top \cdot \otimes_{i=\mu'_1+1}^{\mu'_1+\mu_1}(1 - \mathbf{z}[i], \mathbf{z}[i]). \quad (1)$$

Recall from Sect. 2.1 that, given two vectors with identical tensor structure, their inner product can be computed efficiently (logarithmic in length), from which we have

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

The verification cost for computing $\mathbf{w}_1$ is now reduced 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 $k \ge 2$, 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 can still be applied to $\mathsf{Veloz}_{\mathsf{RS}}$ in each round of iteration to further improve verification efficiency, as unfolded below. Since $\mathbf{z}_k = ((\mathbf{G}_{k-1})_{I_{k-1}} \| \mathbf{w}_{k-1}^\top) \cdot \mathbf{s}_{k-1} \in \mathbb{F}^{2^{\mu_{k-1}}}$ is used to generate $\mathbf{w}_k = (\mathbf{r}_k)_\otimes \cdot \mathbf{Mat}_{2^{\mu'_k} \times 2^{\mu_k}}(\mathbf{z}_k) \in \mathbb{F}^{2^{\mu_k}}$, 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 $\mathbf{G}_i$ is of the form $(1, g_i, \cdots, g_i^{s-1})^\top$. Assuming s is a power of two, $\mathbf{G}_i = ((1, g_i^{\frac{s}{2}}) \otimes (1, g_i^{\frac{s}{4}}) \otimes \cdots \otimes (1, g_i))^\top$. 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.

Now we analyze verification cost for computing $\mathbf{w}_k$. For $k = 1$, it requires $O(\mu'_1)$ multiplications over $\mathbb{F}$. In round $k \ge 2$, 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 $\mathbf{w}_{k-1}^\top$) $\mathbb{F}$-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.

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

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

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

4 Security and Complexity Analysis

4.1 Security

Since our motivation aims at accelerating proof generation, we assume that sub-provers trust each other, and analyze security as in the single-prover setting. We have the following theorem.

Theorem 1. *The PCS construction in Sect. 3.3 is complete, binding, and knowledge sound.*

We sketch our proof here, and present a formal one in Appendix B. Completeness holds since an honest prover always passes verification. To argue binding, we assume it does not hold, and lead to a contradiction based on one-wayness assumption of hash function.

For soundness, we adapt the analysis in Ligerito [32, Section 6.3], where the soundness error is $\epsilon_{\text{RS}} = O\Big(\sum_{k=1}^{\ell-1}\big[(1-\frac{2^{\mu_k}-\rho\cdot 2^{\mu_k}+\rho}{2^{1+\mu_k}})^{|I_k|} + \frac{2^{\mu_k}\cdot\mu_k}{\rho|\mathbb{F}|}\big]\Big)$ for RS code, and $\epsilon_{\text{Lin}} = O\Big(\sum_{k=1}^{\ell-1}\big[(1-\frac{\rho\cdot d_k}{3\cdot 2^{\mu_k}})^{|I_k|} + \frac{d_k\cdot\mu_k}{3|\mathbb{F}|}\big]\Big)$ for Brakedown code. As to knowledge soundness, based on the analysis in [18], we construct an extractor $\mathcal{E}_{\text{in}}$ for the case $\ell = 2$, and then for all $\ell \geq 3$, we build an extractor $\mathcal{E}_{\text{out}}$ repeatedly invoking $\mathcal{E}_{\text{in}}$ from $j = \ell - 1$ to $j = 2$, to extract the desired witness.

4.2 Complexity

Now we analyze the complexity of proving, communication, verification, and proof size in detail, and provide asymptotic summaries of complexity for Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$. 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 $1 \leq k \leq \ell - 1$, each sub-prover $\mathcal{P}_i$ participates in

1. encoding $\mathbf{U}_k \in \mathbb{F}^{2^{\mu'_k}\times 2^{\mu_k}}$: $O(2^{\mu'_k-m}\cdot \mathsf{Enc}_k)$ $\mathbb{F}$, where Enc_k is $O(\mu_k 2^{\mu_k})$ for RS code, and $O(2^{\mu_k})$ for Brakedown code [23];
2. Merkle committing to $\hat{\mathbf{U}}_k \in \mathbb{F}^{2^{\mu'_k}\times\frac{2^{\mu_k}}{\rho}}$: $O(2^{\mu'_k-m}\cdot\frac{2^{\mu_k}}{\rho})$ $\mathbb{H}$ for each $\mathcal{P}_i$, and extra $O(2^m)$ $\mathbb{H}$ for $\mathcal{P}_0$;
3. a μ'_k-round partial SumCheck: for each $\mathcal{P}_i$, the former $\mu'_k - m$ rounds require $O(2^{\mu'_k-m})$ $\mathbb{F}$, while the rest m rounds require $O(m - \lfloor\log(i+1)\rfloor + 1)$ $\mathbb{F}$; and
4. computing $\mathbf{w}_k$ and $\mathbf{u}_k \in \mathbb{F}^{2^{\mu_k}}$: $O(2^{\mu'_k-m+\mu_k})$ $\mathbb{F}$.

So proving costs $\sum_{k=1}^{\ell-1} O(2^{\mu'_k-m}\cdot\mathsf{Enc}_k)$ $\mathbb{F}$ + $\sum_{k=1}^{\ell-1} O(2^{\mu'_k+\mu_k-m})$ $\mathbb{H}$.

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

1. exchanging hash values to commit to $\hat{\mathbf{U}}_k \in \mathbb{F}^{2^{\mu'_k}\times\frac{2^{\mu_k}}{\rho}}$: $O(2^{\mu_k})$ $\mathbb{H}$;
2. sending 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
3. sharing 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. The verifier cost in round $1 \leq k \leq \ell$ mainly consists of

1. $(k \leq \ell - 1)$ running μ'_k rounds of partial SumCheck: $O(\mu'_k)$ $\mathbb{F}$;
2. $(k \leq \ell - 1)$ verifying Merkle paths of $(\hat{\mathbf{U}}_k)_{I_k} \in \mathbb{F}^{2^{\mu'_k}\times|I_k|}$: $O(\lambda\cdot(2^{\mu'_k}+\mu_k))$ $\mathbb{H}$;
3. $(k \leq \ell - 1)$ computing $\mathbf{w}_k \in \mathbb{F}^{2^{\mu_k}}$: $O(\mu'_1 + 2^{\mu_1})$ $\mathbb{F}$ for $k = 1$ and $O(2^{\mu'_k+\mu_k})$ $\mathbb{F}$ for $k \geq 2$ in $\text{Veloz}_{\text{Lin}}$, and $O(\mu'_1)$ $\mathbb{F}$ for $k = 1$ and $O(\lambda\cdot(k-1)\mu'_k)$ $\mathbb{F}$ for $k \geq 2$ in Veloz_{RS};
4. $(k \leq \ell - 2)$ computing $\mathbf{z}_{k+1} \in \mathbb{F}^{2^{\mu_k}}$: $O(\lambda\cdot 2^{\mu_k})$ in $\text{Veloz}_{\text{Lin}}$, and 0 in Veloz_{RS} because (3) already involves $\mathbf{z}_{k+1}$; and
5. verifying $\mathbf{u}_{\ell-1}\cdot(\mathbf{G}_{\ell-1})_{I_{\ell-1}} = (\mathbf{r}_{\ell-1})_\otimes\cdot(\hat{\mathbf{U}}_{\ell-1})_{I_{\ell-1}}$, and $y'_{\ell-1} = \langle\mathbf{w}_{\ell-1}, \mathbf{u}_{\ell-1}\rangle$ in the final round: $O(2^{\mu_{\ell-1}})$ $\mathbb{F}$.

To summarize, verification in $\text{Veloz}_{\text{Lin}}$ 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 Veloz_{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 $1 \leq k \leq \ell - 1$, $\{\mathcal{P}_i\}_{i=0}^{M-1}$ send $\mathcal{V}$ three field elements for SumCheck verification, 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 $O(\sum_{k=1}^{\ell-1}(3\mu'_k + \lambda\cdot 2^{\mu'_k}) + 2^{\mu_{\ell-1}})$ $\mathbb{F}$ + $O(\lambda\cdot\sum_{k=1}^{\ell-1}\mu_k)$ $\mathbb{H}$.

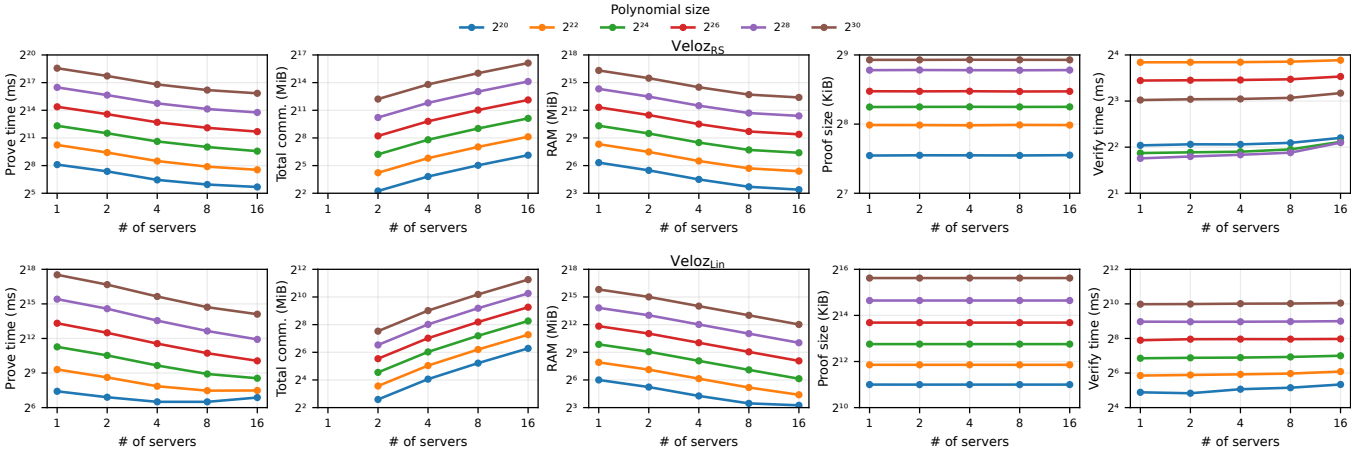

Figure 6: Linear scalability of Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$

Asymptotics for Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$. Now we asymptotically summarize the complexity of Veloz instantiated with RS code and Brakedown code [23], respectively. Denote the size of f by $N = 2^{\mu_0}$.

• Veloz_{RS}. For convenience, we identically set $\mu'_k = \mu'$ for each round number $k \in [1 : \ell]$, so $\mu_k = \mu_0 - k\mu' = \log N - k\mu'$. Observe that the dominant part of proof size is $O\Big(\lambda\big(\ell 2^{\mu'} + \sum_{k=1}^{\ell-1}(\log N - k\mu')\big)\Big) = O\Big(\lambda\ell(2^{\mu'} + \log N + \ell\mu')\Big)$, and that of verification is $O\Big(\lambda\big(\ell^2\mu' + \ell 2^{\mu'} + \sum_{k=1}^{\ell-1}(\log N - k\mu')\big)\Big) = O\Big(\lambda\ell(\ell\mu' + 2^{\mu'} + \log N)\Big)$. Therefore, we set, asymptotically, $2^{\mu'} = \log N = \ell\mu'$, which leads to $\mu' = \log\log N$ and $\ell = \frac{\log N}{\log\log N}$. Then, we obtain prover complexity

$$\sum_{k=1}^{\ell-1} O(2^{\mu'-m} \cdot \frac{N}{2^{k\cdot\mu'}} \cdot \log \frac{N}{2^{k\cdot\mu'}}) = \sum_{k=1}^{\ell-1} O(\frac{N\log N}{M\log^k N} \log \frac{N}{\log^k N})$$
$$= O(\frac{N}{M} \cdot (\frac{\log^2 N}{\log N - 1} - \frac{\log N^2 \cdot \log\log N}{(\log N - 1)^2}))$$
$$= O(\frac{N}{M}\frac{\log^2 N}{(\log N - 1)^2}(\log N - 1 - \log\log N)) = O(\frac{N}{M}\log\frac{N}{\log N}),$$

total communication $O\big(M\cdot\sum_{k=1}^{\ell-1}(\lambda\cdot 2^{\mu'-m} + \frac{N}{(2^{\mu'})^k})\big) = O(\lambda\cdot\frac{\log^2 N}{\log\log N} + M\cdot\frac{N}{\log N})$, and $O(\lambda\cdot\frac{\log^2 N}{\log\log N})$ proof size and verification.

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

• $\text{Veloz}_{\text{Lin}}$. The main difference resides in verification, whose 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 $O(\lambda\cdot(K + \frac{N}{K}))$ verification, and $O(\lambda\cdot(K + \sqrt{\frac{N}{K}}))$ proof size. We note that when $K = \sqrt{N}$ and $K = \sqrt[3]{N}$, verification and proof size reach their minimum, respectively. Therefore, we set $K \in [\sqrt[3]{N}, \sqrt{N}]$, from which we obtain $O(2^{\log K - m}\cdot\frac{N}{K}) = O(\frac{N}{M})$ prover complexity, $O(M\cdot\sum_{k=1}^{\ell-1}(\lambda\cdot 2^{\mu'_k - m} + 2^{\mu_k}) = O(\lambda\cdot(K + \sqrt{\frac{N}{K}}) + M\cdot\frac{N}{K}) = O(\lambda\cdot K + M\cdot\frac{N}{K})$ total communication, $O(\lambda\cdot(K + \frac{N}{K}))$ verification complexity, and $O(\lambda\cdot(K + \sqrt{\frac{N}{K}})) = O(\lambda\cdot K)$ proof size.

5 Implementation and Evaluation

We implement Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ using 6200+ lines of Rust based on the `arkworks` ecosystem [2], where we refer to a Brakedown code implementation[2] for $\text{Veloz}_{\text{Lin}}$. In Sect. 5.1, we evaluate the performance of both schemes on polynomials of sizes $N = 2^{20}$, 2^{22}, 2^{24}, 2^{26}, 2^{28} and 2^{30}. In Sect. 5.2, we compare them to prior code-based distributed PCSs. Since deVirgo [42] and $\text{DePIP}_{\text{FRI}}$ [28] have not provided publicly available implementations, we benchmark Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ against FRIttata[3] and HyperFond[4]. The metrics we consider are proving time, total communication among $\{\mathcal{P}_i\}_{i=0}^{M-1}$, 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 and 256 GiB RAM. For sub-provers $\{\mathcal{P}_i\}_{i=0}^{M-1}$, 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 [32]. Communication among sub-provers in the distributed setting is simulated in a local area network, with 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 provided in Ligerito [32, Section 6.5], where in the first iteration, sub-provers commit to $\hat{\mathbf{U}}_1$ in the subfield $\mathbb{F}_p \subseteq \mathbb{F}_{p^2}$, which results in concretely more efficient proof generation and smaller proof size. The hash function we incorporate in Veloz is BLAKE3[5].

The code rate ρ is set to $\frac{1}{4}$ for Veloz_{RS} and $\frac{3}{5}$ for $\text{Veloz}_{\text{Lin}}$. We set $\lambda = 100$. To guarantee soundness error $\epsilon \le 2^{-100}$ as per the analysis in Sect. 4.1, we set $|I_k| = 148$ in Veloz_{RS} and $|I_k| = 4125$ in $\text{Veloz}_{\text{Lin}}$. During evaluation for each $N \in \{2^{20}, \cdots, 2^{30}\}$, we choose $\mu'_k = \lceil\log\log N\rceil = 5$ in Veloz_{RS}, and $\mu'_k = \frac{\mu_{k-1}}{2}$ in $\text{Veloz}_{\text{Lin}}$.

[2] https://github.com/conroi/lcpc
[3] https://github.com/XuHuaXH/winterfell
[4] https://github.com/n96409816/HyperFond
[5] https://github.com/BLAKE3-team

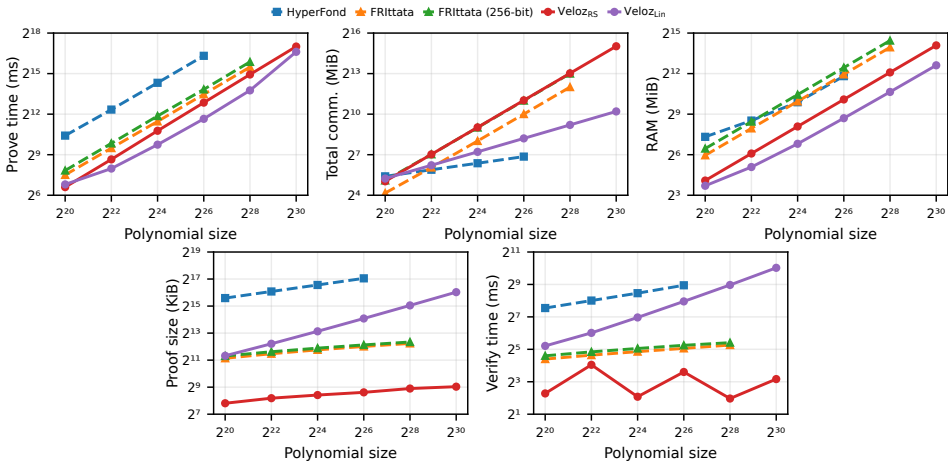

Figure 7: Comparison of Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ to FRIttata and HyperFond

5.1 Linear Scalability

In this section, we evaluate the performance of Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ to show their linear scalability. As shown in Fig. 6, for a fixed polynomial size N, the proving time of both schemes reduce as sub-prover number M increases, which is expected in a distributed system. For a fixed $N = 2^{30}$, the proving time of Veloz_{RS} decreases as M doubles. Specifically, proof generation takes 387s for $M = 1$, while that drops to 74.8s and 58.7s for $M = 8$ and $M = 16$, respectively. These correspond to speedup factors of 5.18 × and 6.60 ×. A similar trend appears in $\text{Veloz}_{\text{Lin}}$. With $N = 2^{30}$, the proving time reduces from 189s ($M = 1$) to 26.9s ($M = 8$) and further to 17.6s ($M = 16$), achieving greater speedup factors of 7.02 × and 10.7 ×. 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. The significance is expected to be better in practical use where more servers are deployed for distribution.

Regarding other metrics under a fixed N, both schemes exhibit the following trends as M increases: the total communication cost grows linearly with M, while memory cost of $\mathcal{P}_0$ decreases. In contrast, the proof size and verification time stabilize, as expected since they are *independent* from M. Concretely, with $N = 2^{30}$ and $M = 16$, the proof size of Veloz_{RS} is 487KB, with only 9.01ms for verification; while those in $\text{Veloz}_{\text{Lin}}$ are 49.2MB and 1.06s, respectively. In conclusion, both schemes achieve remarkable linear scalability.

We remark that an intriguing observation is the abnormal verification time in Veloz_{RS}, where for $N = 2^{22}$ it measures about 14ms, while that of $N = 2^{28}$ is only round 4ms. This essentially stems from our setting for $\mu_{\ell-1}$ in the final round when $\{\mathcal{P}_i\}_{i=0}^{M-1}$ send $\mathbf{u}_{\ell-1}$ directly to $\mathcal{V}$. Concretely, since we fix $\mu'_k = 5$ for each k, the derived $\mu_{\ell-1}$ is 12 for $N = 2^{22}$, and 8 for $N = 2^{28}$. These parameters can be flexibly adjusted to suit practical requirements.

5.2 Comparing to Prior Works

In this subsection, we compare Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ to FRIttata [45] and HyperFond [46], all with 8 sub-provers and tested on the same device. We set $\rho = \frac{1}{4}$ in FRIttata and HyperFond; we set $K = \frac{T}{4}$ (the flexible parameter from Table 1) in FRIttata as in [45, Section 5].

We notice that FRIttata operates on a 256-bit field, while Veloz and HyperFond are implemented on the 128-bit $\mathbb{F}_{p^2}$. For comprehensive presentation as well as alignment, we first evaluate FRIttata's implementation in its original configuration, and focus on benchmarking another evaluation after substituting their field with $\mathbb{F}_{p^2}$ without altering other parameters. To keep fairness in the setting of 128-bit field, we *disable* the subfield technique in the first round of Veloz_{RS} or $\text{Veloz}_{\text{Lin}}$, which concretely increases our proving time and proof size. We also note a methodological difference in sub-prover simulation: FRIttata simulates them sequentially [45, Section 5], so its proving time does not account for communication, while those in HyperFond and Veloz do embody communication costs.

As depicted in Fig. 7 (with $M = 8$), Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ achieve comprehensive performance improvements. Compared to HyperFond, which is communication-efficient, our schemes outperform it significantly in other metrics. Specifically, for $N = 2^{26}$, HyperFond requires 81.6s for proof generation, produces a 133MB proof size, and needs 494ms for verification. In comparison, Veloz_{RS} completes proof generation within 7.40s, achieving a 11.0 × speedup, and yields a compact 383KB proof that verifies in 12.2ms (40.5 × faster). $\text{Veloz}_{\text{Lin}}$ further reduces proving to 3.22s (a 25.3 × speedup over HyperFond), with a 16.9MB proof size verified in 248 ms. Against FRIttata (128-bit), Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ are advantageous in proving time and memory cost. For communication, $\text{Veloz}_{\text{Lin}}$ is the most efficient, followed by FRIttata and then Veloz_{RS}. Regarding proof size and verification time, FRIttata performs between Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$. Concretely for $N = 2^{28}$, FRIttata takes 45.7s to generate a 4.71MB proof, which verifies in 38.4ms. Veloz_{RS} achieves proving in 31.3s (a 1.46 × speedup), producing a shorter 477KB proof that verifies in just 3.91ms (9.82 × faster). $\text{Veloz}_{\text{Lin}}$ offers the fastest proving at 13.9s (a 3.29 × speedup), albeit with a larger 33.1MB proof that takes 501ms to verify. At $N = 2^{30}$, Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ are still capable of proof generation, while the others run out of memory.

In summary, both Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$ enjoy fast proving and efficient memory usage. For other metrics, Veloz_{RS} features smaller proof size and faster verification, while $\text{Veloz}_{\text{Lin}}$ exhibits efficient

communication. We note that the reason behind the "zigzag" verification time of Veloz_{RS} as N doubles is again due to our setting for $\mu_{\ell-1}$, as previously explained in Sect. 5.1.

6 Conclusion and Future Directions

We introduced Veloz, a novel distribution framework for code-based multilinear PCS regardless of code choice, 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 [23], denoted as Veloz_{RS} and $\text{Veloz}_{\text{Lin}}$, respectively. With both featuring transparent setup and plausibly quantum resilience, Veloz_{RS} exhibits efficient verification and tiny proof size, while $\text{Veloz}_{\text{Lin}}$ enjoys linear proving time and field agnosticity. Experimental results demonstrate their remarkable linear scalability. For future works, possible optimization can be exploited to reduce the communication in Veloz_{RS}, and to shorten the proof size in $\text{Veloz}_{\text{Lin}}$. Also, applying both PCSs in distributed SNARKs is an interesting direction.

References

[1] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2087–2104. doi:10.1145/3133956.3134104

[2] arkworks contributors. 2022. `arkworks` *zkSNARK ecosystem.* https://arkworks.rs

[3] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. 2024. 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* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 380–413. doi:10.1007/978-3-031-68403-6_12

[4] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. 2025. WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification. In *Advances in Cryptology – EUROCRYPT 2025*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer Nature Switzerland, Cham, 214–243.

[5] Gal Arnon, Alessandro Chiesa, and Eylon Yogev. 2023. IOPs with Inverse Polynomial Soundness Error. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. 752–761. doi:10.1109/FOCS57990.2023.00050

[6] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 14:1–14:17. doi:10.4230/LIPIcs.ICALP.2018.14

[7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046. https://eprint.iacr.org/2018/046

[8] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. 2017. Interactive Oracle Proofs with Constant Rate and Query Complexity. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 80)*, Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 40:1–40:15. doi:10.4230/LIPIcs.ICALP.2017.40

[9] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986*. Springer-Verlag, Berlin, Heidelberg, 31–60. doi:10.1007/978-3-662-53644-5_2

[10] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. 2019. DEEP-FRI: Sampling Outside the Box Improves Soundness. Cryptology ePrint Archive, Paper 2019/336. https://eprint.iacr.org/2019/336

[11] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. 2020. Linear-Time Arguments with Sublinear Verification from Tensor Codes. In *Theory of Cryptography*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer International Publishing, Cham, 19–46.

[12] Hopwood Sean Bowe, T. George Hornby, and Nathan Wilcox. 2016. Zcash Protocol Specication. https://api.semanticscholar.org/CorpusID:18340182

[13] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. 2025. Blaze: Fast SNARKs from Interleaved RAA Codes. In *Advances in Cryptology – EUROCRYPT 2025*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer Nature Switzerland, Cham, 123–152.

[14] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. 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* (Zagreb, Croatia). Springer-Verlag, Berlin, Heidelberg, 677–706. doi:10.1007/978-3-030-45721-1_24

[15] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *Advances in Cryptology – EUROCRYPT 2023*, Carmit Hazay and Martijn Stam (Eds.). Springer Nature Switzerland, Cham, 499–530.

[16] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. ZKML: An Optimizing System for ML Inference in Zero-Knowledge Proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 560–574. doi:10.1145/3627703.3650088

[17] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020*, Anne Canteaut and Yuval Ishai (Eds.). Springer International Publishing, Cham, 738–768.

[18] Benjamin E. Diamond and Jim Posen. 2024. Proximity Testing with Logarithmic Randomness. *IACR Commun. Cryptol.* 1, 1 (2024).

[19] Amos Fiat and Adi Shamir. 1987. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO' 86*, Andrew M. Odlyzko (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 186–194.

[20] Ethereum Foundation. 2024. "zk-rollups: Scaling solutions for ethereum.

[21] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953. https://eprint.iacr.org/2019/953

[22] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4, Article 27 (Sept. 2015), 64 pages. doi:10.1145/2699436

[23] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. 2023. 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* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 193–226. doi:10.1007/978-3-031-38545-2_7

[24] Yanpei Guo, Xuanming Liu, Kexi Huang, Wenjie Qu, Tianyang Tao, and Jiaheng Zhang. 2025. 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* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 180, 20 pages.

[25] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194.

[26] Jonathan Lee. 2021. Dory: Efficient, Transparent Arguments for Generalised Inner Products and Polynomial Commitments. In *Theory of Cryptography*, Kobbi Nissim and Brent Waters (Eds.). Springer International Publishing, Cham, 1–34.

[27] Chongrong Li, Pengfei Zhu, Yun Li, Cheng Hong, Wenjie Qu, and Jiaheng Zhang. 2025. HyperPianist: Pianist with Linear-Time Prover and Logarithmic Communication Cost . In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 3383–3401. doi:10.1109/SP61157.2025.00202

[28] Weihan Li, Zongyang Zhang, Sherman S. M. Chow, Yanpei Guo, Boyuan Gao, Xuyang Song, Yi Deng, and Jianwei Liu. 2025. Shred-to-Shine Metamorphosis in Polynomial Commitment Evolution. Cryptology ePrint Archive, Paper 2025/1354. https://eprint.iacr.org/2025/1354

[29] Weihan Li, Zongyang Zhang, Yun Li, Pengfei Zhu, Cheng Hong, and Jianwei Liu. 2025. Soloist: Distributed SNARKs for Rank-One Constraint System. Cryptology ePrint Archive, Paper 2025/557. https://eprint.iacr.org/2025/557

[30] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. 2024. Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs. In *2024 IEEE Symposium on Security and Privacy (SP)*. 1777–1793. doi:10.1109/SP54263.2024.00035

[31] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *J. ACM* 39, 4 (oct 1992), 859–868. doi:10.1145/146585.146605

[32] Andrija Novakovic and Guillermo Angeris. 2025. Ligerito: A Small and Concretely Fast Polynomial Commitment Scheme. Cryptology ePrint Archive, Paper 2025/1187. https://eprint.iacr.org/2025/1187

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

[34] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. 2016. Constant-round interactive proofs for delegating computation. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing* (Cambridge, MA, USA) *(STOC*

'16). Association for Computing Machinery, New York, NY, USA, 49–62. doi:10.1145/2897518.2897652

[35] Noga Ron-Zewi and Ron Rothblum. 2024. Local Proofs Approaching the Witness Length. *J. ACM* 71, 3, Article 18 (June 2024), 42 pages. doi:10.1145/3661483

[36] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. 2024. Hekaton: Horizontally-Scalable zkSNARKs Via Proof Aggregation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 929–940. doi:10.1145/3658644.3690282

[37] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. 2013. Interactive proofs of proximity: delegating computation in sublinear time. In *Symposium on the Theory of Computing*. https://api.semanticscholar.org/CorpusID:54211316

[38] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *Advances in Cryptology – CRYPTO 2020*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer International Publishing, Cham, 704–737.

[39] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509. arXiv:https://doi.org/10.1137/S0097539795293172 doi:10.1137/S0097539795293172

[40] Wenhao Wang, Fangyan Shi, Dani Vilardell, and Fan Zhang. 2024. Cirrus: Performant and Accountable Distributed SNARK. Cryptology ePrint Archive, Paper 2024/1873. https://eprint.iacr.org/2024/1873

[41] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. 2018. DIZK: A Distributed Zero Knowledge Proof System. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 675–692. https://www.usenix.org/conference/usenixsecurity18/presentation/wu

[42] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. 2022. zkBridge: Trustless Cross-chain Bridges Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 3003–3017. doi:10.1145/3548606.3560652

[43] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology – CRYPTO 2019*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer International Publishing, Cham, 733–764.

[44] Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2022. 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* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 299–328. doi:10.1007/978-3-031-15985-5_11

[45] Hua Xu, Mariana Gama, Emad Heydari Beni, and Jiayi Kang. 2025. FRIttata: Distributed Proof Generation of FRI-based SNARKs. Cryptology ePrint Archive, Paper 2025/1285. https://eprint.iacr.org/2025/1285

[46] Yuanzhuo Yu, Mengling Liu, Yuncong Zhang, Shi-Feng Sun, Tianyi Ma, Man Ho Au, and Dawu Gu. 2025. HyperFond: A Transparent and Post-Quantum Distributed SNARK with Polylogarithmic Communication. Cryptology ePrint Archive, Paper 2025/1349. https://eprint.iacr.org/2025/1349

[47] Hadas Zeilberger, Binyi Chen, and Ben Fisch. 2024. BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes. In *Advances in Cryptology – CRYPTO 2024*, Leonid Reyzin and Douglas Stebila (Eds.). Springer Nature Switzerland, Cham, 138–169.

A Additional Backgrounds on Proofs and Arguments

PoK and ARK. We adapt the definitions of proof and argument of knowledge (PoK, ARK) from [15]. Recall that an *indexed* relation $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}; \mathbb{w})$, where the index $\mathbb{i}$ is fixed in setup phase. A language $\mathcal{L}(\mathcal{R})$ corresponds to the indexed relation $\mathcal{R}$ such that $(\mathbb{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R})$ if and only if $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$.

Definition 2 (PoK and ARK [15]). *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 $\mathbb{i}$, common input $\mathbb{x}$ and prover witness $\mathbb{w}$, the deterministic indexer $\mathcal{I}$ outputs $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathcal{I}(\mathbb{i})$. 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 $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$,

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

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

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

If the probability above 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 PPT extractor $\mathcal{E}$ such that given oracle access to $\mathcal{A}$,

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

If $\mathcal{A}$ is unbounded, then the ARK 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 $\mathsf{poly}(\lambda, |\mathbb{x}|, o|\mathbb{w}|)$. The interaction of a public-coin ARK can be removed via the Fiat-Shamir transform [19]. 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) [5, 9, 34, 47]). *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 $(\mathbb{x}; \mathbb{w}) \in \mathcal{R}$,

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

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 $\mathcal{A}$,

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

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

Definition 4 (Interactive Oracle Proof of Proximity (IOPP) [37, 47]). *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* $\mathcal{A}$*,*

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

B Proof of Theorem 1

We now prove completeness, binding and knowledge soundness of our construction.

Proof. **Completeness**: we reduce the completeness to that in Ligerito [32]. As in Sect. 3, the evaluation protocol consists of the distributed partial SumCheck and the distributed vector-matrix multiplication, so we analyze completeness for each building block.

In the distributed partial SumCheck, for a single prover $\mathcal{P}$, in round k it sends the verifier $\mathcal{V}$ a polynomial $s_k(x_k)$, which in our distribution is represented as $\sum_{i=0}^{M-1} s_k^{(i)}(x_k)$, with $s_k^{(i)}(x_k)$ locally computed by $\mathcal{P}_i$. Since distribution only alters how $s_k(x_k)$ is obtained, completeness remains unchanged.

The distributed vector-matrix multiplication consists of the following three phases.

(1) Distributed encoding: as in Ligero [1] and Ligerito [32], $\mathbf{U}$ 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 encoded matrix $\hat{\mathbf{U}}$ is close to the prescribed code space 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 $\frac{2^{\mu-\mu'}}{\rho}$ 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, $\mathbf{u} = \mathbf{r}_\otimes \cdot \mathbf{U} = \sum_{i=0}^{M-1} \mathbf{r}_\otimes^{(i)} \cdot \mathbf{U}^{(i)}$, so completeness is clear.

Binding: assume that, given $\mathsf{gp} = (\ell, \{\mu_k', \mu_k, \mathbf{G}_k\}_{k=1}^{\ell-1})$, a PPT adversary $\mathcal{A}$ outputs a commitment cm for $f_0 \neq f_1 \in \mathcal{F}_\mu^{(\le 1)}$ with hints t_0 and t_1, respectively, such that

$$\mathsf{Open}(\mathsf{gp},\mathsf{cm};f_0,t_0) = 1 = \mathsf{Open}(\mathsf{gp},\mathsf{cm};f_1,t_1).$$

Then we have $\mathsf{MC}(t_0) = \mathsf{cm} = \mathsf{MC}(t_1)$ and, denoting $\mathbf{Mat}_{2^{\mu_1'}\times 2^{\mu_1}}(\mathbf{f}_i)$ by $\mathbf{M}_i$ for each i,

$$\Delta_{C_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_i), t_i)) < \frac{\delta_{C_1}}{2}$$

holds for $i = 0, 1$. If $t_0 \neq t_1$, then $\mathcal{A}$ locates a hash collision, which is of negligible probability. Therefore, overwhelmingly, the two inequalities hold under the condition that $t_0 = t_1$. Now, given $f_0 \neq f_1$, there exist two rows with the same index p of $\mathbf{M}_0$ and $\mathbf{M}_1$ that differ, so we obtain two codewords $\mathsf{Enc}_1(\mathbf{M}_0[p,\cdot]) \neq \mathsf{Enc}_1(\mathbf{M}_1[p,\cdot]) \in C_1$, which yields

$$\begin{aligned} &\Delta_{C_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_0), \mathsf{Enc}_1(\mathbf{M}_1)) \\ \ge &\Delta_{C_1}^{1}(\mathsf{Enc}_1(\mathbf{M}_0[p,\cdot]), \mathsf{Enc}_1(\mathbf{M}_1[p,\cdot])) \\ \ge &\delta_{C_1}, \end{aligned}$$

by the definition of relative minimum distance. This, together with the prior condition $t_0 = t_1$ and the following triangle inequality

$$\begin{aligned} \delta_{C_1} &= \frac{\delta_{C_1}}{2} + 0 + \frac{\delta_{C_1}}{2} \\ &> \Delta_{C_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_0), t_0)) + \Delta_{C_1}^{\mu_1'}(t_0, t_1) + \Delta_{C_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_1), t_1)) \\ &\ge \Delta_{C_1}^{\mu_1'}(\mathsf{Enc}_1(\mathbf{M}_0), \mathsf{Enc}_1(\mathbf{M}_1))) \ge \delta_{C_1}, \end{aligned}$$

leads to a contradiction. Hence, we conclude that $f_0 = f_1$.

Soundness: (adapted from [32, Section 6.3]) the soundness error in round $1 \le k \le \ell - 1$ arises from (1) a μ_k'-round distributed partial SumCheck (Protocol 1), (2) a distributed commitment and querying protocol (Protocol 2) for $\mathbf{U}_k \in \mathbb{F}^{2^{\mu_k'} \times 2^{\mu_k}}$, and (3) batching $|I_k| + 1$ columns with $\mathbf{s}_k = (1, \gamma_k, \cdots, \gamma_k^{|I_k|})$.

For (1), by Lemma 1, 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 [23]. 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 $\mathsf{Veloz}_{\mathrm{RS}}$, or $\frac{d\cdot\mu_k}{3|\mathbb{F}|}$ in $\mathsf{Veloz}_{\mathrm{Lin}}$, where u^* is the multilinear polynomial interpolated from $\mathbf{u}^*$.

For (3), by Lemma 1, the error is $\frac{|I_k|+1}{|\mathbb{F}|}$.

In conclusion, conditioned on $\log|\mathbb{F}| = \Omega(\lambda)$, the soundness error is

$$\epsilon_{\mathrm{RS}} = O\Big(\sum_{k=1}^{\ell-1}\Big[(1 - \frac{2^{\mu_k} - \rho\cdot 2^{\mu_k} + \rho}{2^{1+\mu_k}})^{|I_k|} + \frac{2^{\mu_k}\cdot\mu_k}{\rho|\mathbb{F}|}\Big]\Big),$$

in $\mathsf{Veloz}_{\mathrm{RS}}$, and

$$\epsilon_{\mathrm{Lin}} = O\Big(\sum_{k=1}^{\ell-1}\Big[(1 - \frac{\rho\cdot d_k}{3\cdot 2^{\mu_k}})^{|I_k|} + \frac{d_k\cdot\mu_k}{3|\mathbb{F}|}\Big]\Big)$$

in $\mathsf{Veloz}_{\mathrm{Lin}}$.

Knowledge soundness: Recall from Sect. 1.2 that, in the evaluation protocol of [18], the prover $\mathcal{P}$ sends $\mathbf{u}$ directly to the verifier $\mathcal{V}$, for verifying that $\mathsf{r}_\otimes \cdot \hat{\mathbf{U}}_I = \mathbf{u} \cdot \hat{\mathbf{G}}_I$, and that $y_1' = \langle \mathbf{u}, \mathbf{w}\rangle$, which, essentially, is the special case in [32] by taking the iteration number $\ell = 2$. It has been proved in [18] that the case $\ell = 2$ is knowledge sound, as captured by the following Lemma

Lemma 2 ((adapted) Knowledge soundness [18, Theorem 5]). *Taking* $\ell = 2$*, for any* $\mu \in \mathbb{N}$ *and any PPT algorithm* $\mathcal{A}$*, there exists an expected polynomial-time extractor* $\mathcal{E}$*, such that the following probability*

$$\Pr\left[\begin{array}{c|c} \mathsf{Eval}(\mathsf{gp},\mathbb{x};\perp)\langle\mathcal{A}_2(\mathbf{st}),\mathcal{V}\rangle & \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda,\mu) \\ = 1 \wedge (\mathbb{x};\mathbb{w}) \notin \mathcal{R}_{\mathsf{Eval},\mathsf{gp}} & (\mathbb{x} = (\mathsf{cm},\mathbf{z},y),\mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ & \mathbb{w} = (f,t) \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{gp},\mathbb{x}) \end{array}\right]$$

is $\mathsf{negl}(\lambda)$.

Now we inductively prove that Ligerito [32] and hence our Veloz satisfy knowledge soundness.

Denote by $\mathcal{E}_{\text{in}}$ the extractor in Lemma 2. Assuming now that $\ell \geq 3$, we construct the extractor $\mathcal{E}_{\text{out}}$ that extracts $f \in \mathcal{F}_{\mu_0}^{(\leq 1)}$ and $t = \hat{\mathbf{U}}_1$ as follows. Given a PCS evaluation instance $(\mathsf{cm}, \mathbf{z}, y)$, and the access to the adversary $\mathcal{A}$, $\mathcal{E}_{\text{out}}$ observes the messages $\{\mathsf{cm}_k = \mathsf{MC}(\hat{\mathbf{U}}_k)\}_{k=1}^{\ell-1}$, randomness $\mathbf{r}_1 \| \cdots \| \mathbf{r}_{\ell-1}$, and the vector $\mathbf{u}_{\ell-1}$ directly sent to $\mathcal{V}$ in the final round. It then invokes $\mathcal{E}_{\text{in}}$ on instance

$$(\mathsf{cm}_{\ell-1}, \mathbf{r}_{\ell-1} \| \mathbf{z}[-\mu_{\ell-1} :] \in \mathbb{F}^{\mu'_{\ell-1}+\mu_{\ell-1}}, \langle \mathbf{u}_{\ell-1}, \mathbf{z}[-\mu_{\ell-1} :]_{\otimes} \rangle),$$

and extracts $\mathbf{U}_{\ell-1}$ and $\hat{\mathbf{U}}_{\ell-1}$. Note that $\mathbf{U}_{\ell-1}$ corresponds to the multilinear polynomial $f(\mathbf{r}_1, \cdots, \mathbf{r}_{\ell-2}, \cdot) \in \mathcal{F}^{(\leq 1)}_{\mu'_{\ell-1}+\mu_{\ell-1}} = \mathcal{F}^{(\leq 1)}_{\mu_{\ell-2}}$.

Recall that $\mathbf{U}_{\ell-1} = \mathbf{Mat}_{2^{\mu'_{\ell-1}} \times 2^{\mu_{\ell-1}}}(\mathbf{u}_{\ell-2})$, so $\mathcal{E}_{\text{out}}$ flattens $\mathbf{U}_{\ell-1}$ for $\mathbf{u}_{\ell-2}$, and invokes $\mathcal{E}_{\text{in}}$ on instance

$$(\mathsf{cm}_{\ell-2}, \mathbf{r}_{\ell-2} \| \mathbf{z}[-\mu_{\ell-2} :] \in \mathbb{F}^{\mu'_{\ell-2}+\mu_{\ell-2}}, \langle \mathbf{u}_{\ell-2}, \mathbf{z}[-\mu_{\ell-2} :]_{\otimes} \rangle)$$

and extracts $\mathbf{U}_{\ell-2}$ and $\hat{\mathbf{U}}_{\ell-2}$. Again, $\mathbf{U}_{\ell-2}$ corresponds to the multilinear polynomial $f(\mathbf{r}_1, \cdots, \mathbf{r}_{\ell-3}, \cdot) \in \mathcal{F}^{(\leq 1)}_{\mu'_{\ell-2}+\mu_{\ell-2}} = \mathcal{F}^{(\leq 1)}_{\mu_{\ell-3}}$.

$\mathcal{E}_{\text{out}}$ continues by iteratively invoking $\mathcal{E}_{\text{in}}$ on instance

$$(\mathsf{cm}_k, \mathbf{r}_k \| \mathbf{z}[-\mu_k :] \in \mathbb{F}^{\mu'_k+\mu_k}, \langle \mathbf{u}_k, \mathbf{z}[-\mu_k :]_{\otimes} \rangle)$$

to extract $\mathbf{U}_k$ and $\hat{\mathbf{U}}_k$, for $\ell - 3 \geq k \geq 1$. $\mathcal{E}_{\text{out}}$ finally obtains $\mathbf{U}_1$ and its encoding $\hat{\mathbf{U}}_1$, or equivalently, $f(\cdot) = u_0(\cdot) \in \mathcal{F}^{(\leq 1)}_{\mu_0}$ and $t = \hat{\mathbf{U}}_1$, as desired. □