Shred-to-Shine Metamorphosis in Polynomial Commitment Evolution

Weihan Li[*] Zongyang Zhang[†] Sherman S. M. Chow[‡] Yanpei Guo[§]
Boyuan Gao[¶] Xuyang Song[‖] Yi Deng[**] Jianwei Liu[¶]

Abstract

Polynomial commitment schemes (PCSs) enable verifying evaluations of committed polynomials. Multilinear (ML) PCSs from linear codes are favored for their prover time. Distributed MLPCSs further reduce it by enabling multiple provers to distribute both commitment and proof generation.

We propose PIP_{FRI}, an FRI-based MLPCS that unites the linear prover time of PCSs from encodable codes with the compact proofs and fast verification of Reed–Solomon (RS) PCSs. By cutting FFT and hash overhead for both committing and opening, PIP_{FRI} runs 10× faster in prover than the RS-based DeepFold (Usenix Security' 25) while retaining competitive proof size and verifier time, and beats Orion (Crypto '22) from linear codes by 3.5-fold in prover speed while reducing proof size and verification time by 15-fold.

Its distributed version $\text{DEPIP}_{\text{FRI}}$ delivers the first code-based distributed SNARK for arbitrary circuits over single multilinear polynomials, and further achieves accountability. $\text{DEPIP}_{\text{FRI}}$ outperforms DeVirgo (CCS '22)—the only prior code-based distributed MLPCS, limited to data-parallel circuits and lacking accountability—by 25× in prover time and 7× in communication, with the same number of provers.

A central insight in both constructions is the shred-to-shine technique. It further yields a group-based MLPCS of independent interest, with 16× shorter structured reference string and 10× faster opening time than multilinear KZG (TCC'13).

1 Introduction

Polynomial commitment schemes (PCSs) [32, 36] let a prover commit to a polynomial f in μ variables with degree bound d and later prove that $f(\boldsymbol{x}) = y$ at a public point $\boldsymbol{x}$ without revealing f. Succinct non-interactive arguments of knowledge (SNARKs) [19, 30, 47, 55] use PCSs with the proof size and verification cost sublinear in the polynomial size d^{μ}. Modern SNARK constructions follow the "PIOP (polynomial interactive oracle proof) [20] + PCS" framework [19, 20, 30], so the overall efficiency depends on the PIOP and PCS.

Multilinear PCSs (MLPCSs) [39, 51] for polynomials with $d = 2$ and $\mu > 1$ pair naturally with linear-prover-time multilinear PIOPs [19, 47] for SNARKs. MLPCSs from linear codes [32, 34, 59, 60] operate directly over finite fields, avoiding more costly group operations, and pushing prover times even lower. These PCSs have been applied in machine learning [2, 42] and scalable blockchain [46, 48], protecting billions of dollars. In these domains, prover time is the key scalability metric, while proof size and verifier time remain vital for blockchains, deciding the verification circuit size and thus the prover time of the recursive SNARK [33].

However, current code-based MLPCSs can not simultaneously achieve *linear (and concretely efficient) prover time* along with *poly-log (and concretely efficient) proof sizes and verifier times*. Specifically, RS-code MLPCSs [34, 58, 60] provide the latter with several hundred KBs proofs, but the prover is quasi-linear, optimal with RS-encoding via Fast Fourier Transformers (FFTs). By contrast, MLPCSs from linear-time encodable codes attain the former but either yield square-root proof size and verifier cost [32], or several-MBs proofs and concretely 10× slower verifier time [56].

Distributed PCSs [40, 41, 52, 54] offer another way to reduce the prover time by enabling multiple provers to perform committing and opening collaboratively. They are also required for constructing distributed SNARKs using the "distributed PIOP [40, 41] + distributed PCS" paradigm. Distributed PIOPs are information-theoretic and compatible with all PCSs from different cryptographic primitives. Using code-based distributed PCSs to build prover-efficient distributed SNARKs is a promising direction due to their original efficiency.

A distributed PCS exhibits *full linear speedup* if: 1) with ℓ provers, each prover's work is $1/\ell$ of the single-prover

[*]School of Cyber Science and Technology, Beihang University. Part of the work was done while visiting The Chinese University of Hong Kong.
[†]Corresponding author. Sch. of Cyber Sci. & Tech., Beihang University.
[‡]Dept. of Information Engineering, Chinese University of Hong Kong.
[§]National University of Singapore.
[¶]School of Cyber Science and Technology, Beihang University.
[‖]Anoma.
[**]Institute of Information Engineering, Chinese Academy of Sciences.

work while 2) the proof size and verifier time remain unchanged. DeVirgo [54], the only code-based distributed PCS (also an MLPCS), achieves full linear speedup in theory but has four drawbacks. Its base PCS, Virgo [59], has quasi-linear opening complexity and is 10× slower in prover than recent non-distributed FRI-based PCSs [34, 60], so DeVirgo needs at least 8 provers to match their prover times, which increases deployment costs. Secondly, unlike non-code-based distributed MLPCSs [40, 52], DeVirgo only distributes work when each sub-prover handles a *separate, unrelated* polynomial; it cannot split the workload of a single, interdependent polynomial across provers. This limitation prevents its use in distributed SNARKs for general circuits, as state-of-the-art prover-efficient schemes [40, 52] use distributed multilinear PIOPs, which require distributed PCSs supporting single and linear-size polynomials. Thirdly, unlike some non-code-based distributed PCSs [41, 52], DeVirgo lacks accountability: an honest master prover cannot detect malicious sub-provers, making the overall system vulnerable to adversaries. Finally, the concrete amortized communication can reach hundreds of MBs, making it impractical for low-bandwidth networks.

Such a state of affairs motivates us to ask two questions:

1. Can we achieve a code-based MLPCS with *more efficient* prover as well as *affordable* proof size and verifier time?
2. Can we design an *accountable* code-based distributed PCS supporting *single* polynomials with *improved efficiency*?

1.1 Our Contribution

PIP: Shred-to-Shine MLPCS-to-MLPCS Upgrade. We propose PIP, a simple and general MLPCS framework built from MLPCS that achieves faster prover with at most slight increases in proof size and verification. We instantiate PIP with code-based and group-based MLPCSs, obtaining two PCSs, PIP_{FRI} and PIP_{KZG}, respectively, with strong trade-offs. PIP also inspires to achieve generality for distributed PCSs.

PIP_{FRI}: Efficient FRI-based MLPCS with zero knowledge. Compared with FRI-based MLPCSs (*e.g.*, PolyFRIM [60], DeepFold [34]), PIP_{FRI} reduces FFT-based committing costs from $O(N \log N)$ to $O(N \log m)$ by splitting N into $m\ell$ for tunable m. It also cuts hash-based committing and opening costs from $O(N)$ to $O(m)$ using smaller Merkle trees. Its opening cost stays at $O(N)$, but requires only a single linear combination ($\mathbb{F}_{\text{rlc}}$), which is practically efficient. Table 1 shows the main trade-offs, and Table 2 reports micro-benchmarks with 2× and 5× speedups in FFTs and Merkle trees. Experiments show that despite the complexity gap, PIP_{FRI} is the first FRI-based PCS achieving faster prover time than PCSs from linear-time codes over RS-friendly fields (*e.g.*, Orion) due to the more efficient recursive-PCS construction of PIP.

We also equip PIP_{FRI} with zero knowledge, a feature essential for zk-SNARKs and their applications, which is missing from several recent works [6, 19, 58, 60] (Table 1).

$\text{DEPIP}_{\text{FRI}}$: Accountable and general distributed PCS with improved computation and reduced communication. We propose $\text{DEPIP}_{\text{FRI}}$, a code-based distributed PCS with linear speedup (cf. Table 3). It also adds the following *new* features.

1. $\text{DEPIP}_{\text{FRI}}$ is the first code-based scheme with **accountability**. Some group-based distributed PCSs [41, 52] support accountability readily as each sub-proof is generated by a sub-prover locally, which allows the master prover to verify directly. Code-based schemes offer no analogous mechanism: each sub-proof is generated by multiple sub-provers. We propose new designs to support accountability.
2. $\text{DEPIP}_{\text{FRI}}$ is **general**, supporting the distribution of both single polynomials and multiple independent polynomials. The former leads to the first code-based distributed SNARK for **arbitrary** circuits. Currently, such scheme [54] could achieve linear prover speedup only by splitting independent polynomials, as current distributed FFTs [53] fail to handle interdependent workloads efficiently (*cf.*, Section 1.2).
3. The amortized **computation** and **communication** costs for *openings* are $O(m/\ell)$, an ℓ-fold gain over DeVirgo. This breaks the apparent $O(m)$ complexity, achieved by our shred-to-shine method in the non-distributed PIP_{FRI}. It first splits a size-$O(m\ell)$ polynomial into ℓ size-$O(m)$ polynomials, and merges these ℓ polynomials into one of the same size m. As a result, in the distributed setting, the ℓ provers, though each holding a size-m polynomial, can handle a *single* size-m polynomial collaboratively. For this scenario, we introduce a new distributed folded polynomial computation approach, optimally achieving the $O(m/\ell)$ bound. This approach also improves the opening complexity of distributed FRI, which is a standalone advance as FRI is the core sub-protocol in code-based schemes [6, 34]. It could support other FRI-based distributed PCSs.

Empirical Evaluation. We implement PIP_{FRI}, $\text{DEPIP}_{\text{FRI}}$, and (distributed) SNARKs from them. PIP_{FRI} strikes a strong balance among code-based PCSs, especially on prover time. It has a 10× faster prover (commit + open) time than FRI-based DeepFold, and 3.5× faster prover than Orion from linear-time codes, while maintaining the proof size and verifier time from competitive to 15× better. Also, PIP_{FRI} is memory-efficient among code-based PCSs with poly-log proof size, which is 10× smaller than DeepFold, 5× smaller than Orion, and firstly supports size-2^{29} polynomials on 32 GB RAM. Overall, PIP_{FRI} leads to SNARKs with 2-10× faster provers than DeepFold and 10-20× smaller proof sizes than Orion.

$\text{DEPIP}_{\text{FRI}}$ achieves *full* linear speedup, in contrast to DeDory [40] that achieves linear speedup while sacrificing proof size and verification (from $O(\log m\ell)$ to $O(\log m^2\ell)$). Apart from generality and accountability, $\text{DEPIP}_{\text{FRI}}$ cuts the prover time by 25× and communication overhead by 7× compared to DeVirgo, thanks to our new distributed FRI. The proof size and verifier time are competitive. The generality further makes $\text{DEPIP}_{\text{FRI}}$ lead to the first code-based distributed

Table 1: Comparisons of size-$N = m\ell$ code-based MLPCSs with λ-bit security

Scheme	Commit	Open	Verify	Proof size	zk
Brakedown [32]	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\sqrt{N})\,\mathbb{F}/\mathsf{H}$	$O(\lambda\sqrt{N})\,\mathbb{F}+O(\log N)\,\mathsf{H}$	○
Orion [56]	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\sqrt{N})\,\mathbb{F}/\mathsf{H}$, no preprocessing	$O(\lambda\log^2 N)\,\mathsf{H}$	◐
HyperPlonk [19]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	○
Virgo [59]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	●
PolyFRIM [60]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	○
Basefold [58]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	○
DeepFold [34]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\log^2 N)\,\mathsf{H}$	◐
PIP$_{\mathbf{FRI}}$	$O(N\log m)\,\mathbb{F}, O(m)\,\mathsf{H}, O(m)\,\mathsf{H}_\ell$	$O(N)\,\mathbb{F}_{\mathsf{rlc}}, O(m)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log m+\lambda\ell)\,\mathbb{F}, O(\log^2 m)\,\mathsf{H}$	$O(\lambda\log m+\lambda\ell)\,\mathbb{F}, O(\log^2 m)\,\mathsf{H}$	●

$\mathbb{F}$ is a field with large multiplicative cosets, and H a hash function. Both denote operation time or size, as context requires. H_ℓ: hashing $O(\ell)$ entries (versus H for a constant number); $\mathbb{F}_{\mathsf{rlc}}$: one-time random linear combination over $\mathbb{F}$; implementations use $\log N \le \ell < \log^2 N$, $m = N/\ell$. ○: No zero-knowledge (zk) variant explicitly provided; ◐: No zk implementation provided; ●: Both are given.

Table 2: Micro-benchmarks of PolyFRIM and PIP$_{\text{FRI}}$ for size-2^{20} polynomials over a 64-bit finite field

Type	Term	Time	Term	Time
FFT	$O(N\log N)\,\mathbb{F}$	620 ms	$O(N\log m)\,\mathbb{F}$	315 ms
Build MT	$O(N)\,\mathsf{H}$	2,258 ms	$O(m)\,\mathsf{H}, O(m)\,\mathsf{H}_\ell$	468 ms
RLC	-	-	$O(N)\,\mathbb{F}_{\mathsf{rlc}}$	52 ms

*MT: Merkle tree; RLC: random linear combination over a field

SNARK for arbitrary circuits, with a 4× faster prover time and a 5× faster verifier time than HyperPianist [40], a group-based distributed SNARK with time efficiency.

PIP$_{\text{KZG}}$: Group-based PCS with shorter SRS and faster time. We instantiate PIP to build PIP$_{\text{KZG}}$ from mKZG [45] (*cf.*, Table 4). PIP$_{\text{KZG}}$ reduces the size of structured reference string (SRS) as well as the opening and verifier complexities. Compared with *univariate* Bünz *et al.* [17] with sublinear-size SRS, PIP$_{\text{KZG}}$ is secure in the algebraic group model (AGM) instead of the more abstract generic group model (GGM) [29].

Experiments show that PIP$_{\text{KZG}}$ has 16× shorter SRS, 2× faster prover, and 2× faster verifier time than mKZG, but sacrifices the proof size by 1.8× (<3KB for most cases). Compared with the trustless group-based Dory [39], PIP$_{\text{KZG}}$ requires trusted setups, but is more efficient due to operating over type-1 groups and eliminating the expensive operations over target groups. Concretely, PIP$_{\text{KZG}}$ has a 2× faster prover, 4× faster verifier, and 10× smaller proof size.

1.2 Technical Overview

Construction of PIP. The high-level idea of PIP is "shred-to-shine": shredding a large multilinear polynomial into smaller sub-polynomials, handling each sub-polynomial efficiently, eliding expensive operations, and regrouping proofs to let performance shine by batching. Technically, we apply *tensor products* to recast the validation of a multilinear polynomial of size $N = m\ell$ as ℓ checks on size-m sub-polynomials. Observing that these sub-polynomials are evaluated at the same point, we use *batch PCS* to slash prover workload, delivering a speedup by reducing the number of expensive operations for the evaluation proof, exactly where performance matters, while asymptotic improvement in the total cost would likely require new paradigm-specific techniques.

In other words, PIP offers a simpler design, shining through different PCS frameworks to reduce cost-dominating operations, covering FFTs, Merkle trees, and group operations. Instantiating PIP produces PCSs with more compact proofs and lower prover and verifier cost when compared with previous code-based attempts, *e.g.*, those adopting recursive-proof techniques [10, 32, 56]. See Section 3.1 for details.

Construction of PIP$_{\text{FRI}}$. We build PIP$_{\text{FRI}}$ by instantiating PIP with PolyFRIM [60], which is less efficient than the latest DeepFold [34]. Recall that PIP is general and compatible with any FRI-based PCS. That PIP$_{\text{FRI}}$ surpasses DeepFold underscores the transformative impact of PIP. To build PIP$_{\text{FRI}}$, we face two challenges: (1) A direct instantiation requires building $O(\ell)$ separate Merkle trees for $O(\ell)$ sub-polynomials in the committing phase, undermining performance gains. (2) Sending sub-polynomial evaluations to the verifier leaks extra information even if PolyFRIM were zero-knowledge.

For challenge (1), by analyzing the structural patterns of opened entries in PolyFRIM, we aggregate entries likely opened together into a single leaf. This insight enables PIP$_{\text{FRI}}$ to use a *single* Merkle tree in the commitment phase, reducing the prover hash costs from $O(\ell\cdot m)\,\mathsf{H}$ into $O(m)\,\mathsf{H}_\ell + O(m)\,\mathsf{H}$.

For challenge (2), we firstly build zk-PolyFRIM by merging the tensor-product foldings inherent to PolyFRIM with a masking coefficient strategy from DeepFold. Observing that it suffices to prove the validity of one target evaluation other than all sub-evaluations in batch PCS, we then exploit the linearity of RS codes to build the virtual target polynomial while not sending the sub-evaluations. We prove the soundness using the proximity gap theorem for RS codes [8].

Construction of DEPIP$_{\text{FRI}}$. The challenge to build *general* FRI-based distributed PCSs lies in the lack of **efficient dis-**

Table 3: Distributed MLPCSs with ℓ provers supporting only ℓ size-m parallel polynomials or a single size-$m\ell$ polynomial as well

Scheme	Trans.	Poly	$\mathcal{P}_i$: Commit	$\mathcal{P}_i$: Open	Comm.: Commit	Comm.: Open	$\mathcal{V}$ & $\lvert\pi\rvert$	Acct.	Circuit
DemKZG [40, 52]	no	single	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(\log m\ell)$	$O(\log m\ell)$	$O(\log m\ell)$	yes	arbitrary
DeDory [40]	yes	single	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(\log m\ell)$	$O(\log m\ell)$	$O(\log m^2\ell)$	no	arbitrary
DeVirgo [54]	yes	parallel	$O(m\log m)$	$O(m)$	$O(m)$	$O(m)$	$O(\lambda\ell+\log^2 m)$	no	data-parallel
DEPIP$_{\text{FRI}}$	**yes**	**single**	$O(m\log m)$	$\mathbf{O(m/\ell)}$	$O(m)$	$\mathbf{O(m/\ell)}$	$O(\lambda\ell+\log^2 m)$	**yes**	**arbitrary**

Trans.: transparent. $\mathcal{P}_i/\mathcal{V}$: sub-prover/verifier complexity over a field. Note that $O(m)$ group multi-scalar exponentiations, as in DemKZG and DeDory, equal to $O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$ over a field where $\log\lvert\mathbb{F}\rvert=\omega(\log m)$ [32]. $\lvert\pi\rvert$: proof size. Comm.: amortized communication complexity. Acct.: accountability. Circuit: supported SNARK circuit type.

Table 4: Group-based MLPCSs for polynomial of size $N=m\ell$

Scheme	SRS size	Commit	Open	Verifier	Proof size
mKZG	$O(N)\,\mathbb{G}$	$O(N)\,\mathbb{G}$	$O(N)\,\mathbb{G}$	$O(\log N)P$	$O(\log N)\,\mathbb{G}$
PIP$_{\text{KZG}}$	$O(m)\,\mathbb{G}$	$O(N)\,\mathbb{G}$	$O(N)\mathbb{G}_{\mathsf{rlc}}$, $O(m)\,\mathbb{G}$	$O(\ell)\,\mathbb{G}$, $O(\log m)\,P$	$O(\ell)\,\mathbb{G}$, $O(\log m)\,\mathbb{G}$

$\mathbb{G}$: operations over group. $\mathbb{G}_{\mathsf{rlc}}$: linear combination between group and field. P: pairing operations. Efficiency: $\mathbb{G}_{\mathsf{rlc}}>\mathbb{G}>P$.

tributed FFT. Most FRI-based PCSs require linear-size FFTs. However, existing scheme in DIZK [53] on a size-$m\ell$ vector across ℓ provers costs $O(m\log^2 m)$ per prover, increasing the overall prover time. This differs from group-based schemes, where the multi-scalar exponentiations are naturally distribution-friendly [1, 43]. We achieve generality by our "shred-to-shine" method in PIP$_{\text{FRI}}$. The size-$m\ell$ target polynomial is split into ℓ size-m sub-polynomials, and the FFT for the target polynomial is transformed into ℓ *independent* sub-FFTs for sub-polynomials. As a result, sub-provers can run sub-FFTs locally, taking amortized $O(m\log m)$ optimally.

We next improve the **opening** phase of DEPIP$_{\text{FRI}}$. Suppose in the i-th round, provers compute a codeword $f_i|_{L_i}$ over a domain L_i, corresponding to the i-th folded degree-$O(|L_i|)$ polynomial f_i. In DeVirgo, each sub-prover $\mathcal{P}_j$ holds a size-$O(2\cdot|L_i|)$ sub-polynomial $f_{i-1}^{(j)}$ in the $(i-1)$-th round, computes the degree-$O(|L_i|)$ sub-polynomial $f_i^{(j)}$ and its codeword, and then exchanges with other provers to build f_i and $f_i|_{L_i}$. $\mathcal{P}_j$ would compute and send all entries of $f_i^{(j)}|_{L_i}$ to other provers, and hence, the amortized prover complexity and communication complexity are both $O(|L_i|)$.

We improve the efficiency via the following observation: optimally, $\mathcal{P}_j$ only needs the j-th part of $f_i|_{L_i}$, which is of length $O(|L_i|/\ell)$ given ℓ provers. We then build a *distributed folded polynomial computation method* to compute $f_i|_{L_i}$ collaboratively and directly, instead of computing $f_i^{(j)}|_{L_i}$ separately and then combining them. As a result, the amortized prover and communication complexities are both reduced to $O(|L_i|/\ell)$. Equipped with the distributed FFT and our improved distributed FRI, we build DEPIP$_{\text{FRI}}$ over PIP$_{\text{FRI}}$.

Finally, we achieve **accountability** for DEPIP$_{\text{FRI}}$. Known accountable distributed PCSs [52] are group-based: sub-provers send to the master prover $\mathcal{P}_0$ (prover-)independent sub-proofs, and $\mathcal{P}_0$ combines homomorphically into the final proof. Each sub-proof corresponds naturally to a public sub-relation, allowing $\mathcal{P}_0$ to verify and detect malicious behavior easily. In contrast, proofs in code-based distributed PCSs (DEPIP$_{\text{FRI}}$ included) are jointly generated by multiple provers. The malice that arose in a sub-prover's sub-proof may actually originate from others, complicating accountability.

To resolve this, we let sub-provers additionally generate sub-proofs for their sub-polynomials, and verify them via PCS verification. By the construction of DEPIP$_{\text{FRI}}$, these sub-proofs deterministically relate to the jointly-generated proof, and can serve as an authentic reference relying on the soundness. $\mathcal{P}_0$ can reconstruct and compare the correct DEPIP$_{\text{FRI}}$'s proof with the received one. Any discrepancy *pinpoints the exact* malicious sub-prover. This adds only a minor overhead: $\mathcal{P}_0$'s cost rises from $O(\ell\log m)$ to $O(\ell\log^2 m)$, still well below each sub-prover's $O(m\log m)$ cost, where $m\gg\ell$.

1.3 Related Work

Code-based PCSs all rely on a low-degree test (LDT) to verify that a committed vector is within a constant relative Hamming distance of a valid codeword. Standard LDTs include: 1) the direct LDT [3] employed in PCSs [11, 32, 56] from linear-time encodable codes; 2) FRI-(like)-based LDTs for RS codes [7, 8, 58] as used in PCSs [34, 58–60]. The direct LDT applies to all constant-relative-distance linear codes but yields a proof size linear in the vector length. In contrast, FRI offers poly-log proof size, though limited to RS codes.

PCSs from direct LDT. Brakedown [32] introduces a practical generalized Spielman (GS) code and an MLPCS with a linear-time prover relying on the direct LDT from Ligero [3]. It offers square-root verifier complexity and proof sizes, reaching tens of MBs. Orion [56] uses a code-based SNARK [59] to recursively prove the verification circuit of Brakedown, reducing the proof size to poly-log. However, the proofs

remain several MBs and the verifier complexity stays square-root without trusted preprocessing. Block *et al.* [11] modify the GS code in Brakedown into an expand-accumulate (EA) code with larger code distance and hence smaller proof size, while leaving overall complexities unchanged. Blaze [14] uses the repeat-accumulate-accumulate (RAA) code to tailor an MLPCS. Currently, it only works over binary fields.

FRI-based PCSs generally have higher prover complexity and slower prover times than linear-prover PCSs. Surprisingly, our FRI-based PIP_{FRI} is the first FRI-based PCS to achieve better prover efficiency than state-of-the-art linear-prover PCSs (*e.g.*, Orion) over RS-friendly prime fields.

PCS from FRI. The univariate FRI-PCS [50] with linear opening complexity is not directly compatible with multilinear polynomials. Zeromorph [38], a general MLPCS-from-univariate-PCS approach, requires quasi-linear openings when instantiated with FRI-PCS. Virgo [59] and HyperPlonk [19] build MLPCSs with $O(N \log N)$ openings due to committing size-$O(N)$ auxiliary polynomials.

PolyFRIM [60] refines FRI to bypass this committing overhead, firstly achieving $O(N)$ openings. Basefold [58] exploits FRI as a multilinear polynomial oracle and combines it with the multilinear sum-check [31], resulting in an MLPCS with linear opening complexity. However, this oracle-based approach weakens soundness, necessitating extra verifier queries and resulting in proofs near 1 MB in practice.[1] DeepFold [34] improves Basefold's soundness. It has a concretely faster prover time and smaller proof size than PolyFRIM. Notably, only Virgo and DeepFold offer zero-knowledge variants.

Recent STIR [5] and WHIR [6] are drop-in replacements of FRI with fewer queries and verifier's hashes. Like FRI-based MLPCSs mentioned above, replacing FRI as STIR or WHIR leads to schemes with better proof size and verifier time. However, STIR and WHIR do not improve the prover.

Compared with these PCSs, PIP_{FRI} achieves faster prover complexity and prover time in committing and opening, as well as zero knowledge. PIP_{FRI} also features competitive concrete proof size and verifier time with DeepFold (hence outperforms PCSs from direct LDT), though DeepFold originally outperforms the sub-protocol PolyFRIM used in PIP_{FRI}.

Distributed PCSs enable multiple sub-provers, each handling a sub-polynomial, to jointly produce commitments and evaluation proofs. Multiple distributed PCSs exist for distinct polynomial forms from different primitives. Existing distributed MLPCSs include group-based distributed mKZG and distributed Dory [40], which can distribute single polynomials. In contrast, DeVirgo [54], the sole code-based distributed (ML)PCS, supports only multiple independent polynomials. When used for SNARKs this would limit its usage to only data-parallel circuits as current linear-prover distributed PIOPs for general circuits [40] involve a single linear-size polynomial.

[1] Basefold offers an "FRI-like" LDT for foldable linear codes, which generalizes RS codes. PCS with this LDT is field-agnostic but less efficient.

Table 5: Efficiency of distributed FRIs in Xu *et al.* and ours

	$\mathcal{P}_i$	$\mathcal{P}_0$	$\mathcal{V}/\pi$	Total Comm.
[57]	$O(m \log m)^*$	$O(m\ell)$	$O(\ell \log m + \log^2 m)$	$O(m\ell)$
Ours	$O(m \log m)$	$O(\ell \log m)$	$O(\lambda\ell + \log^2 m)$	$O(m\ell)$

*Xu *et al.* **assume** that $\mathcal{P}_i$ initially holds the polynomial evaluations, hence not needing FFTs. Ours starts from FFTs and features a quasi-linear complexity. Operations other than FFTs also cost $O(m)$ time.

Table 6: Performance of distributed FRIs for size-2^{25} polynomials in Xu *et al.* and ours, with 4 or 16 provers ($\#\mathcal{P}_i$)

	$\#\mathcal{P}_i$	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Total Comm.
[57]	4	30	10	700	1.07 GB
Ours	4	9.7	3.8	283	1.50 GB
[57]	16	10	30	1,500	1.07 GB
Ours	16	2.6	3.8	271	1.90 GB

Typically, distributed PCSs assume all sub-provers are honest. Recent works [41, 52] consider situations where some sub-provers can be malicious and generate fake sub-proofs. Accountability guarantees that if some sub-prover is malicious to generate fake sub-proofs, the honest master prover can detect it. Existing accountable PCSs are all group-based, and extending the techniques into code-based schemes is not natural, as mentioned in Section 1.2. Differently, $\text{DEPIP}_{\text{FRI}}$ is the first code-based accountable and general distributed PCS.

Group-based PCSs start from univariate KZG [36]. Its multilinear version, mKZG [45], requires logarithmic proof size and verification in polynomial size. (m)KZG requires a linear-size SRS, which can be hundreds of GBs for large-scale computations [53] and affect the discrete-logarithm assumption [35]. An exception is Bünz *et al.* [17] with $O(\sqrt{N})$ SRS, but its security relies on GGM [44], a model generally regarded as stronger than AGM [29] assumed in (m)KZG-based SNARKs. Dory [39] eliminates the trusted setups, also with a sublinear common-reference-string. However, its prover and verifier operate over the target group instead of the type-one group as in mKZG, which is reportedly 4× worse [39].

Concurrent Work. Recently, Xu *et al.* [57] propose a "fold-and-batch" distributed FRI, sharing our sub-goal of devising distributed code-based PCS, while mainly considering empirical communication saving. On a high level,

Note that Xu *et al.* do not achieve full linear speedup, with verifier time and proof size worsening as the number of provers grows. Our prover efficiency comes from the improved distributed opening with $O(m/\ell)$ sub-prover and amortized communication, which could also accelerate theirs.

We build distributed code-based PCSs and distributed SNARKs for general circuits using different methodologies. Xu *et al.* design a bivariate PCS paired with the bivariate PIOP in Pianist [41]. We build an MLPCS and use the multilinear

PIOP in HyperPianist [40], which is out of their reach due to incompatibility. Our faster FRI and linear-time PIOP yield a faster prover in distributed PCS and SNARK.[2] Fast proving is exactly why we chose to study and improve code-based PCS.

Apart from efficiency, accountability is also not in their scope. In addition, their distributed PCS can only distribute bivariate polynomials like $f(X,Y)=\sum_i f_i(X)L_i(Y)$, assigning *independent* polynomials $f_i(X)$ to provers. In contrast, our shred-to-shine method enables distributed FFTs and supports (single) univariate, bivariate, and multilinear polynomials. Finally, our shred-to-shine method also speeds up the prover of *non-distributed* FRI-PCS by 10× (Table 8). Its benefits also shine through other PCSs, such as Xu *et al.*'s and mKZG.

2 Preliminaries

$[n]$ denotes $\{1,\ldots,n\}$ for integer n. Lowercases like $f,\hat{f}$, and $\tilde{f}$ represent polynomials on finite field $\mathbb{F}$. We use $\tilde{f}$ and $\hat{f}$ to specifically denote multilinear and its twist univariate polynomials. Bold denotes vectors on $\mathbb{F}$. x_i denotes the i-th entry of $\boldsymbol{x}$. $\langle \boldsymbol{a},\boldsymbol{b}\rangle$ and $\boldsymbol{a}\otimes\boldsymbol{b}$ denote the inner and tensor product of $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. $\otimes_{i=1}^{n}\boldsymbol{x_i}$ means $\boldsymbol{x_1}\otimes\cdots\otimes\boldsymbol{x_n}$. $N=d^\mu$ represents polynomial size for μ-variate polynomials with degree bound d. Given a coset L and code rate $\rho\in(0,1)$, RS code $\mathsf{RS}[L,\rho]\in\mathbb{F}^{|L|}$ means $\{f|_L|\deg(f)<\rho|L|\}$, and $f|_L$ is the evaluation of polynomial f with size $\rho|L|$ on L.

Merkle tree is a vector commitment of a size-N vector with $O(N)$ prover time, $O(\log N)$ verifier time, and $O(\log N)$ proof size. It comprises three algorithms. $\mathsf{rt}\leftarrow\mathsf{MT.Commit}(\boldsymbol{v})$ outputs the Merkle root for vector $\boldsymbol{v}$. Given a query set I, $(\{v_i\}_{i\in I},\mathsf{path})\leftarrow\mathsf{MT.Open}(\mathsf{rt},I,\boldsymbol{v})$ outputs v_i (the queries) and path (the verification path). $\{0,1\}\leftarrow\mathsf{MT.Verify}(\mathsf{rt},I,\{v_i\}_{i\in I},\mathsf{path})$ verifies the validity by checking the consistency among $\{v_i\}$, rt, and path. We use Merkle trees built by collision-resistant and non-invertible hash functions.

Argument of Knowledge (AoK). An interactive argument for NP relation $\mathcal{R}$ allows a prover to convince a verifier that there exists a witness $\mathtt{w}$ s.t. $(\mathtt{x},\mathtt{w})\in\mathcal{R}$ for statement $\mathtt{x}$. AoK additionally requires knowledge soundness, *i.e.*, $\mathtt{w}$ is efficiently extractable. Appendix A.1 recalls the definitions.

Polynomial Commitment Scheme (PCS). A PCS for polynomial f is defined by the algorithms or protocols below [32].

- $\mathsf{pp}\leftarrow\mathsf{Gen}(1^\lambda,d,\mu)$: takes security parameter λ, variate μ, and variate-degree bound d; generates public parameter pp.
- $C\leftarrow\mathsf{Com}(\mathsf{pp},f)$: takes f and outputs commitment C.
- $b\leftarrow\mathsf{VerPoly}(\mathsf{pp},C,f,aux)$: verifies the opening of C given possible auxiliary input aux; outputs $b\in\{0,1\}$ at the end.
- $b\leftarrow\mathsf{Eval}(\mathsf{pp},C,\boldsymbol{x},y;f)$ is an (interactive) argument. Prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold commitment C, point $\boldsymbol{x}$, and evaluation y. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that C commits f of bounded size d^μ and $f(\boldsymbol{x})=y$. $\mathcal{V}$ finally outputs $b\in\{0,1\}$.

[2] Xu *et al.* did not report figures for their distributed PCSs and SNARKs.

PCSs satisfy completeness, polynomial binding, and knowledge soundness. Zk-PCSs additionally satisfy honest-verifier zero knowledge. Appendix A.2 recalls formal definitions.

HyperPlonk and PolyFRIM. HyperPlonk [19, §B] runs an FRI and an FRI-like scheme. Given a size-$N=2^\mu$ multilinear polynomial $\tilde{f}_0$, the prover folds the twisted univariate polynomial f_0 to obtain and commit to $f_1,\ldots,f_{\mu+1}$, where the size of $f_i(X)$ is $N/2^i$. This is similar to FRI, except that the folding parameter is an entry of the evaluation point $\boldsymbol{x}$ rather than a random challenge. In the i-th round ($i\in[\mu]$), $f_i(X)=g_{i-1}(X)+x_i\cdot h_{i-1}(X)$, where g_{i-1},h_{i-1} satisfy $f_{i-1}(X)=g_{i-1}(X^2)+X\cdot h_{i-1}(X^2)$. The prover and verifier then run a batch FRI-PCS to show the low-degree properties of $f_1,\ldots,f_{\mu+1}$ and their consistency.

The batch FRI-PCS requires padding polynomials $f_1,\ldots,f_{\mu+1}$ to the same size N, leading to a quasi-linear opening complexity. PolyFRIM proposes a *rolling batch FRI* specified for LDTs of these polynomials, observing that the j-th round folded-polynomial of f_1 is evaluated over the same domain as f_{j+1}, and hence can be combined directly. This process can apply recursively, reducing the quasi-linear opening complexity to linear, which is detailed in Appendix A.

3 PIP: Shred-to-Shine MLPCS from MLPCS

This section proposes PIP, a shred-to-shine general MLPCS framework built on and improving MLPCS. Its goal is to reduce expensive operations in PCSs, exemplified by FFTs, group multi-scalar exponentiations (both costing super-linear complexities over fields) and Merkle-tree constructions. To achieve this, we transform the evaluation validity of a multilinear polynomial $\tilde{f}$ of size $N=m\ell$ into ℓ independent checks on size-m sub-polynomials. We split its coefficient $(f_1,f_2,\ldots,f_N)$ into ℓ parts of size m. For $j\in[\ell]$, we define a sub-polynomial $\tilde{f}_j$ with coefficient $(f_{(j-1)\cdot m+1},\ldots,f_{j\cdot m})$.

Without loss of generality (w.l.o.g.), assume m and ℓ are power-of-two integers. Suppose the evaluation point is $\boldsymbol{x}=(x_1,\ldots,x_{\log m\ell})$. We define two public vectors $\boldsymbol{v}=\otimes_{k\in[\log m]}(1,x_k)$ and $\boldsymbol{w}=\otimes_{k\in[\log\ell]}(1,x_{k+\log m})$. Observe that

$$y=\langle(\tilde{f}_1(\boldsymbol{v}),\ldots,\tilde{f}_\ell(\boldsymbol{v})),\boldsymbol{w}\rangle. \tag{1}$$

This forms our key insight guiding the following construction. Instead of committing $\tilde{f}$ directly, the prover commits to each $\tilde{f}_1,\ldots,\tilde{f}_\ell$ separately, sends claimed evaluations $(\tilde{f}_1(\boldsymbol{v}),\ldots,\tilde{f}_\ell(\boldsymbol{v}))$, and proves their validity. Once those proofs pass, the verifier checks if Equation (1) holds with public $\boldsymbol{w}$.

As $\tilde{f}_1,\ldots,\tilde{f}_\ell$ are evaluated at the same point $\boldsymbol{v}$, our framework naturally leverages batch PCS (Definition 3.1), which allows efficient simultaneous opening of multiple polynomials at a common point. Batch PCS is available for nearly all leading schemes from different cryptographic primitives, such as KZG [36], Bulletproofs [15], Dark [16], and FRI-PCS [50].

Protocol 1 (PIP). *Let* $\mathsf{PC}_b = (\mathsf{Gen}_b, \mathsf{Com}_b, \mathsf{VerPoly}_b, \mathsf{Eval}_b)$ *be the algorithms of a batch MLPCS.*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$. pp *includes* $(\mathbb{F}, \mu, m, \ell, \mathsf{pp}_b)$, *where* $\mathsf{pp}_b = \mathsf{Gen}_b(1^\lambda, 2, \log m)$ *and* $m\ell = 2^\mu$.
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f})$. *Given* $\tilde{f}$ *and its coefficient* $\boldsymbol{f}$, $\mathcal{P}$ *does:*
  1. *Partition* $\tilde{f}$ *into* ℓ *sets of size-*m *polynomials* $\{\tilde{f}_j\}_{j\in[\ell]}$, *where the coefficients of* $\tilde{f}_j$ *are* $(f_{(j-1)\cdot m+1}, \ldots, f_{j\cdot m})$.
  2. $Run C_j \leftarrow \mathsf{Com}_b(\mathsf{pp}_b, \tilde{f}_j)$ *for* $j \in [\ell]$. *Output* $(C_1, \ldots, C_\ell)$.
- $b \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, \tilde{f})$. $\mathcal{V}$ *parses* $\tilde{f}$ *and runs* $b_j \leftarrow \mathsf{VerPoly}_b(\mathsf{pp}_b, C_j, \tilde{f}_j)$. *Output* 1 *iff* $b_j = 1$ *for* $j \in [\ell]$.
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; \tilde{f})$. *Given* $\boldsymbol{x}$ *and* $y = \tilde{f}(\boldsymbol{x})$,
  1. $\mathcal{P}$ *and* $\mathcal{V}$: *compute public vectors* $\boldsymbol{v} = \otimes_{k\in[\log m]}(1, x_k)$ *and* $\boldsymbol{w} = \otimes_{k\in[\log \ell]}(1, x_{k+\log m})$.
  2. $\mathcal{P} \to \mathcal{V}$: $\boldsymbol{u} = \tilde{f}_1(x_1, \ldots, x_{\log m}), \ldots, \tilde{f}_\ell(x_1, \ldots, x_{\log m})$.
  3. $\mathcal{P}$ *and* $\mathcal{V}$: *run the batch evaluation protocol* Eval_b *to prove that* $\tilde{f}_j(x_1, \ldots, x_k) = u_j$ *for* $j \in [\ell]$.
  4. $\mathcal{V}$: *accepts iff* Eval_b *outputs* 1 *and* $\langle \boldsymbol{u}, \boldsymbol{w} \rangle = y$.

In fact, the batch evaluation protocol is a simpler version of the linear combination scheme in Halo-infinite [12].

Definition 3.1 (Batch PCS). *Suppose a PCS for a size-*$O(N)$ *polynomial with* $t_\mathcal{P}(N)$ *opening time,* $t_\mathcal{V}(N)$ *verifier time, and* $\pi(N)$ *proof size. It is a batch PCS if, given* ℓ *sets of {commitment, evaluation point, evaluation}, i.e.,* $\{C_j, \boldsymbol{x_j}, y_j\}_{j\in[\ell]}$, *there exists an AoK for the following relation* $\mathcal{R}_{\mathsf{Eval}}$:

$$(\{C_j, \boldsymbol{x_j}, y_j\}; \{f_j\}_{j\in[\ell]}) : f_j(\boldsymbol{x_j}) = y_j \wedge \mathsf{VerPoly}(\mathsf{pp}, C_j, f_j) = 1.$$

Moreover, the opening time is $O(\ell) + t_\mathcal{P}(N)$, *the verifier time is* $O(\ell) + t_\mathcal{V}(N)$, *and the proof size is* $O(\ell) + \pi(N)$.

Protocol 1 yields the concrete construction of PIP. Theorem 3.1 shows its advantages.

Theorem 3.1 (PIP: Shred-to-shine MLPCS). *Protocol 1 is an MLPCS for size-*$O(N)$ *polynomials that satisfies:*

- *Committing is* parallelizable-friendly, *which can be divided into* ℓ independent *committings of size-*m *polynomials;*
- *The opening time is* $O(N) + t_\mathcal{P}(m)$;
- *The verifier time is* $O(\ell) + t_\mathcal{V}(m)$;
- *The proof size is* $O(\ell) + \pi(m)$.

PIP *especially shines in the following cases:*

- *If the original committing time* $t_\mathsf{C}(N)$ *is* $O(N \log N)$, *then the committing complexity is reduced to* $O(N \log m)$.
- *If* $t_\mathcal{P}(N) = O(N \log N)$, *the opening complexity would be* $O(N)$ *by setting* $m = O(N/\log N)$ *and* $\ell = O(\log N)$.
- *If both* $t_\mathcal{V}(N)$ *and* $\pi(N)$ *are* $O(\log^c N)$ *for some constant* $c \geq 1$, *then by setting* $m = O(N/\log^c N)$ *and* $\ell = O(\log^c N)$, *neither the verifier complexity nor the proof size increases.*
- *If the original PCS needs a size-*$O(N)$ *SRS for trusted setups, the SRS size of* PIP *would be reduced to* $O(m)$.

Appendix B gives the detailed proofs, where we first prove Equation (1), and then the security properties of PIP.

Extending to the Univariate Case. Protocol 1 is designed for multilinear polynomials and can be adapted for univariate ones. Note that $(1, x, \ldots, x^{N-1})$ for any x can be expressed as a tensor product $\otimes_{i=1}^{\log N}(1, x^{2^{i-1}})$. For size-$N$ univariate $f(X)$, we can form an equivalent multilinear $\tilde{f}(X_1, \ldots, X_\mu)$ by setting $X_1 = X, X_2 = X^2, \ldots, X_i = X^{2^{i-1}}, X_\mu = X^{N/2}$.

Adding Zero Knowledge. Protocol 1 is not zero-knowledge because $\boldsymbol{u}$ in Step 2 of Eval leaks information about $\tilde{f}$.

We present a general zero-knowledge transformation for univariate polynomials $\hat{f}$. W.l.o.g., let its size be $N = m\ell$. Write $\hat{f}(X) = \sum_{i=1}^{\ell} X^{m(i-1)} \hat{f}_i(X)$. The prover chooses random $r_1, \ldots, r_{\ell-1}$, and defines masked polynomials $\hat{f}'_{i\in[\ell-1]}$, where $r_0 = r_\ell = 0$ and $\hat{f}'_i(X) = \hat{f}_i(X) + r_i X^m - r_{i-1}$. The prover commits to each $\hat{f}'_i$ and sends masked vector $\boldsymbol{u}' = (\hat{f}'_1(\alpha), \ldots, \hat{f}'_\ell(\alpha))$ instead. Correctness holds for every α, as $\sum_{i=1}^{\ell} X^{m(i-1)} \hat{f}_i(X) = \sum_{i=1}^{\ell} X^{m(i-1)} \hat{f}'_i(X)$ and hence $\hat{f}(\alpha) = \langle \boldsymbol{u}', (1, \alpha^m, \ldots, \alpha^{m(\ell-1)}) \rangle$. Degree in each $\hat{f}'_i$ increases by 1, and this can be proven directly by LDTs [7] or PCS-based LDTs [18] with minimal overhead.

Section 4 presents a zero-knowledge method specifically for code-based MLPCSs. Generalizing it to all MLPCSs is left for future work.

3.1 Comparisons with Other PCSs

Notable PCS breakthroughs pursue algorithmic improvements, including the group-based Hyrax [51] and the code-based Brakedown [32], Orion [56], and Ligero++ [10]; yet PIP stands out in simplicity, generality, and efficiency.

Comparison with Hyrax. Hyrax arranges the N coefficients of $\tilde{f}$ into an $m \times m$ matrix U ($m = O(\sqrt{N})$) and evaluates it as $\boldsymbol{b}U\boldsymbol{a}^\top$ for public $\boldsymbol{a}, \boldsymbol{b}$. The prover commits each row of U via group-based vector commitments, enabling the verifier to homomorphically derive a commitment to $\boldsymbol{c} = \boldsymbol{b}U$. The prover then proves $\langle \boldsymbol{c}, \boldsymbol{a} \rangle$ via an inner product argument (IPA) [15].

In PIP, $\boldsymbol{w}$ and $\boldsymbol{v}$ play the roles of $\boldsymbol{b}$ and $\boldsymbol{a}$. PIP cleverly exploits the tensor-product structure of vector $\boldsymbol{v}$, making it feasible to firstly handle the computation of $U\boldsymbol{w}^\top$ rather than the conventional way of $\boldsymbol{v}U$. This avoids the need for commitment homomorphism, making it cover code-based PCSs.

Instantiating PIP with the ordinary-group-based PCS [13, 15] recovers Hyrax [51], highlighting PIP's expressiveness.

Comparison with Brakedown. Brakedown shares a similar matrix arrangement and handles $\boldsymbol{c} = \boldsymbol{b}U$ before $\langle \boldsymbol{c}, \boldsymbol{a} \rangle$ like Hyrax. The prover encodes each row of U using linear codes, and proves each row is (close to) a valid codeword via a direct LDT. Direct LDT guarantees that querying several columns of U suffices to check the validity of $\langle \boldsymbol{c}, \boldsymbol{a} \rangle$. It makes the proof

size include $\boldsymbol{c}$ and several columns of U, which is $O(\sqrt{N})$. The $O(\sqrt{N})$ verifier cost is similar. Direct LDT is crucial to achieve succinctness, but optimally only square-root.

PIP removes the need for row encoding and direct LDTs. Instantiated with FRI-based PCSs, PIP can lead to PCSs with poly-log proof size and verifier complexity.

Comparison with Orion and Ligero++. Orion and Ligero++ build recursive proofs upon Brakedown. Specifically, Orion employs an FRI-based recursive SNARK [59] to prove the size-$O(\sqrt{N})$ verification circuit of Brakedown. The recursive SNARK has a proof size poly-log in the circuit size, leading to an $O(\log^2 N)$ proof size. However, using the SNARK requires encoding the columns of the matrix U, leading to double-dimension codewords. Opening Merkle trees for such codewords leads to several-MB concrete proof size [19].

PIP adopts "recursive PCSs" rather than the recursive SNARKs, eliminating the need for large SNARK verification circuits that specify *randomly* queried columns [19] of the encoded matrices. Again, PIP removes the need for row encoding and direct LDTs, which avoids the complex handling of double-dimension codeword Merkle trees. This yields a much smaller proof, reducing the size to hundred KBs.

Ligero++, a SNARK for general circuits, can be modified into a PCS because polynomial evaluation is a kind of linear constraint. Ligero++ arranges the witness into a matrix U, encodes each row of U, and then transforms the linear constraint validity into the inner product relations of a public vector and several columns of U. The inner product relations are proved by an FRI-based IPA Virgo [59]. Similar to Brakedown, Ligero++ also needs double-dimension codeword Merkle trees, leading to large concrete proof size as well. Also, Ligero++ uses IPAs as recursive proofs, while we use recursive PCSs, unlocking possibilities of using more efficient FRI-based sub-PCSs instead of the sub-IPA Virgo. The prover time gap can be larger than 10× [60].

Overall, PIP is simple and general for not needing extra assumptions or tools such as IPAs, commitment homomorphism, or specialized encodings. It relies solely on a batch PCS, a standard component in almost all existing PCSs [12]. Further, PIP leads to more efficient PCSs due to recursive PCSs. These advantages lie in our exploitation of the tensor-product properties of $\boldsymbol{v}$, which enables us to first handle $U\boldsymbol{v}$ as sub-polynomial evaluations.

3.2 PIP_{KZG}: mKZG with Shorter SRS

Using mKZG as PC_b in Protocol 1, we obtain PIP_{KZG}. Let $\tilde{f}$ be a size-$m\ell$ multilinear polynomial partitioned into ℓ sub-polynomials $\tilde{f}^{(1)},\ldots,\tilde{f}^{(\ell)}$, each of size m. Thanks to mKZG's homomorphism, $\mathcal{P}$ can send only the commitments $\mathsf{Com}(\tilde{f}^{(1)}),\ldots,\mathsf{Com}(\tilde{f}^{(\ell)})$, and avoids sending the sub-polynomial evaluations $\tilde{f}^{(1)}(\boldsymbol{x}),\ldots,\tilde{f}^{(\ell)}(\boldsymbol{x})$. $\mathcal{V}$ constructs a virtual commitment to $\tilde{f}$ by $\prod_{i\in[\ell]}\mathsf{Com}(\tilde{f}^{(i)})^{w_i}$.

Appendix C presents the formal scheme and proof. Our implementation of PIP_{KZG} (Appendix G.2) shows that it has a 16-30× shorter SRS, 10× faster opening time, and 1.3× faster verifier time than mKZG, with 2× larger but concretely 3 KB proof size for size-2^{24} polynomials. Combining PIP_{KZG} with the HyperPlonk PIOP [19] yields a SNARK with 16-30× shorter SRS than combining mKZG, while other metrics remain competitive (with <10% gaps).

4 PIP_{FRI}: Prover-Efficient FRI-based PCS

This section builds PIP_{FRI} by combining PIP and PolyFRIM. We choose it due to the linear openings and smaller proof size than Basefold. The instantiation leverages batch and zero-knowledge PolyFRIM, both representing new contributions. We first present them, and then build PIP_{FRI}.

4.1 Batch and Zero-Knowledge PolyFRIM

Batch PolyFRIM. Given ℓ multilinear polynomials $\tilde{f}^{(1)},\ldots,\tilde{f}^{(\ell)}$, our goal is to efficiently prove $\tilde{f}_j(\boldsymbol{x})=y_j$ for $j\in[\ell]$. We observe that in the i-th "FRI-like" round, the ℓ folded polynomials $\hat{f}_i^{(1)},\ldots,\hat{f}_i^{(\ell)}$ are produced with the same folding parameter x_i. By the linearity of RS code, we transform separated openings of $\{\tilde{f}^{(j)}(\boldsymbol{x})\}_{j\in[\ell]}$ into opening $\sum_{j=1}^{\ell}\gamma^j\tilde{f}^{(j)}(\boldsymbol{x})$ once for a random challenge γ. If $\tilde{f}^{(j)}(\boldsymbol{x})\neq y_j$ for some j, the probability of $\sum_{j=1}^{\ell}\gamma^j\tilde{f}^{(j)}(\boldsymbol{x})=\sum_{j=1}^{\ell}\gamma^{j-1}y_j$ is bounded by $\ell/|\mathbb{F}|$. Also, PolyFRIM requires proving the low-degree properties of twist univariate polynomials $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$. This is achieved by the batch FRI [8]: If $\sum_{j=1}^{\ell}\gamma^j\hat{f}^{(j)}$ is low-degree, then with high probability each $\hat{f}^{(j)}$ is low-degree.

A direct batch PolyFRIM leads to the $O((\lambda+\ell)\log N+\log^2 N)$ proof size and verification. The term $O(\ell\log N)$ stems from the prover opening ℓ Merkle trees for size-$O(N)$ codewords of $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ and the verifier checking these trees in the first round. This worsens the proof size and verification.

We resolve this with an optimized Merkle tree commitment, denoted by $\mathsf{MT.Com}([\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0}])$, that groups together entries opened simultaneously into a single Merkle tree leaf. Let the committed codewords be $\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0}$. By the design of PolyFRIM and FRI, if the i-th entry of $\hat{f}^{(j)}|_{L_0}$ is queried for some $j\in[\ell]$, the i-th entries in $\hat{f}^{(j)}|_{L_0}$ for all $j\in[\ell]$ will be queried. Conversely, if a particular entry in some $\hat{f}^{(j)}|_{L_0}$ is not queried, then it is not queried in the others either. This allows the prover to pack the same-location entries from all codewords into a single Merkle tree leaf. With this approach, a single Merkle tree suffices, and both the proof size and verifier time are reduced to $O(\lambda\log N+\ell+\log^2 N)$.

Zero-knowledge PolyFRIM. For a size-N multilinear polynomial $\tilde{f}$, two places in PolyFRIM leak knowledge: 1) in the first round, queries to $\hat{f}$ leak information; 2) from the second round, queries to $\hat{f}$'s folded polynomials leak information.

For 1), we mask $\tilde{f}$ as $\tilde{f}'$ without affecting completeness. We adopt the techniques in DeepFold [34] to add random $(r_1,\ldots,r_N)$ to $\tilde{f}$ with coefficient $(f_1,\ldots,f_N)$. With this padding, $\tilde{f}(\boldsymbol{x}) = \tilde{f}'(\boldsymbol{x},0)$ for $\boldsymbol{x} = (x_1,\ldots,x_\mu)$. As PolyFRIM processes this *public* vector one variable by one in each round, we can add an extra round to handle the variable 0. For 2), we introduce a size-$2N$ random masking polynomial $\tilde{s}$. The prover commits to $\tilde{f}'$ and $\tilde{s}$, claims that $y = \tilde{f}(\boldsymbol{x})$ and $y_s = \tilde{s}(\boldsymbol{x})$, and runs PolyFRIM with $\hat{f}' + \alpha \cdot \hat{s}$ for a verifier-chosen random α. At the end, the verifier checks if the evaluation is $y + \alpha \cdot y_s$. It is zero-knowledge as: 1) in the first round, every opened evaluation on $\hat{f}'$ is randomized. 2) from the second round, $\hat{s}$ ensures no knowledge leakage.

We use batch zk-PolyFRIM as a sub-protocol to build PIP_{FRI}. Due to page limitations, we omit the formal scheme.

4.2 Construction of ZK-PIP_{FRI}

This section presents zk-PIP_{FRI}. Equipped with batch zk-PolyFRIM, we can build a PCS following Protocol 1. However, such a direct combination fails to achieve zero knowledge because the sub-polynomial evaluations $\boldsymbol{u}$ in Protocol 1 are related to the $\tilde{f}$'s coefficient, which leaks knowledge.

To resolve this, we note that, unlike the batch scheme, PIP_{FRI} does not require proving all these sub-polynomial evaluations but only the target evaluation. We rely on the linearity of RS codes to construct a virtual target polynomial and a virtual evaluation while not sending $\boldsymbol{u}$. Specifically, given the target polynomial $\tilde{f}$ and the split sub-polynomials $\tilde{f}_1,\ldots,\tilde{f}_\ell$ as in Protocol 1, we modify each $\tilde{f}_j$ into $\tilde{f}^{(j)}$ by adding random m elements, leading to $\tilde{f}^{(1)},\ldots,\tilde{f}^{(\ell)}$.

Let $\boldsymbol{x}' = (x_1,\ldots,x_k)$ and $\boldsymbol{w} = \otimes_{k\in[\log \ell]}(1, x_{k+\log m})$. By the properties of Protocol 1 and zk-PolyFRIM, we have $\tilde{f}(\boldsymbol{x}) = \sum_{j=1}^{\ell} w_j \cdot \tilde{f}^{(j)}(\boldsymbol{x}',0)$. As $\boldsymbol{w}$ is public, given the query access to $\tilde{f}^{(1)},\ldots,\tilde{f}^{(\ell)}$, the prover and verifier can construct a virtual polynomial $\tilde{f}' = \sum_{j=1}^{\ell} w_j \cdot \tilde{f}^{(j)}$, and invoke the zk-PolyFRIM to prove the validity of the evaluation $\tilde{f}'(\boldsymbol{x}',0) = \tilde{f}(\boldsymbol{x})$.

Protocol 2 presents the formal algorithms for zk-PIP_{FRI}. In Step 3, prover $\mathcal{P}$ computes and commits the "FRI-like" folded polynomials $\hat{f}_1,\ldots,\hat{f}_{\sigma+1}$ following zk-PolyFRIM, taking the input multilinear polynomial as $\sum_{j=1}^{\ell} w_j \tilde{f}^{(j)} + \alpha\tilde{s}$ and its univariate twist polynomial. In Step 4, the verifier checks the consistency of these folded polynomials. In Step 5, the verifier checks the low-degree properties of folded polynomials and input polynomials $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$. This is done via a combination of the rolling batch FRI for folded polynomials and the batch FRI for input polynomials.

Theorem 4.1. *Protocol 2 is a zk-PCS with $O(N\log m)\mathbb{F} + O(m)\mathsf{H}_\ell + O(m)\mathsf{H}$ committing complexity, $O(N)\mathbb{F}_{\mathsf{rlc}} + O(m)\mathbb{F}/\mathsf{H}$ opening complexity, and $O(\lambda\log m + \lambda\ell + \log^2 m)$ verifier complexity and proof size, or $O(\lambda\log m + \lambda\ell)\mathbb{F} + O(\log^2 m)\mathsf{H} + O(\lambda)\mathsf{H}_\ell$ in detail.*

Complexity. We analyze the complexities below.

Protocol 2 (zk-PIP_{FRI}).

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$: *pp includes $\mathbb{F}$, $\ell = O(\log N)$, $m = 2^\mu/\ell$, and a size-$O(m)$ multiplicative coset L_0.*
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f})$: *Given $\tilde{f}$ as input, the prover $\mathcal{P}$ does:*
  1. *Split $\tilde{f}$'s coefficients into ℓ size-m vectors, as in Protocol 1. Add m random entries to each. Denote these size-$2m$ univariate twist polynomials by $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$.*
  2. *Pick a random size-$2m$ polynomial $\hat{s}$. Compute $C \leftarrow \mathsf{MT.Com}([\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0},\hat{s}|_{L_0}])$ obtained via the optimized Merkle tree commitment in Section 4.1.*
- $b \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, \hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0},\hat{s}|_{L_0},\tilde{f})$: *Decode $\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0}$ to obtain $\hat{f}_1,\ldots,\hat{f}_\ell$. Output 1 iff $C = \mathsf{MT.Com}([\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0},\hat{s}|_{L_0}]) \wedge \tilde{f} = \sum_{j\in[\ell]} w_j \tilde{f}^{(j)}$.*
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y, y_s; (\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)},\hat{s}))$: *Let $\sigma = \log m$.*
  1. *$\mathcal{P} \to \mathcal{V}$: claimed evaluations $y = \tilde{f}(\boldsymbol{x})$ and $y_s = \tilde{s}(\boldsymbol{x})$.*
  2. *$\mathcal{V} \to \mathcal{P}$: a random challenge $\alpha \in \mathbb{F}$ for zero knowledge.*
  3. *$\mathcal{P} \to \mathcal{V}$: $\{C_i\}_{i\in[\sigma]}, \hat{f}_{\sigma+1}$. $\mathcal{P}$ sets $\hat{f}_0 \leftarrow \sum_{j\in[\ell]} w_j \cdot \hat{f}^{(j)} + \alpha\hat{s}$. Compute $\hat{f}_i(X) \leftarrow \hat{g}_{i-1}(X) + x_i \cdot \hat{h}_{i-1}(X)$ for $i \in [\sigma+1]$, where $x_{\sigma+1} = 0$. $\hat{g}_{i-1}(X)$ and $\hat{h}_{i-1}(X)$ are obtained by uniquely decomposing $\hat{f}_{i-1}(X)$ as $\hat{g}_{i-1}(X^2) + X \cdot \hat{h}_{i-1}(X^2)$. For $i \in [\sigma]$, $\mathcal{P}$ computes $C_i \leftarrow \mathsf{MT.Com}(\hat{f}_i|_{L_i})$, where $L_i = \{x^2 : x \in L_{i-1}\}$. When $i = \sigma+1$, $\hat{f}_{\sigma+1}$ is a constant that must equal $y + \alpha y_s$.*
  4. ***Consistency check:** Repeat below for $q = O(\lambda)$ times:*
     - *a. $\mathcal{V} \to \mathcal{P}$: a random challenge $\beta \in \mathbb{F}$.*
     - *b. $\mathcal{P}$: invokes $\mathsf{MT.Open}$ to open $\{\hat{f}_i(\pm\beta^{2^i})\}_{i\in[0,\,\sigma]}$.*
     - *c. $\mathcal{V}$: verifies Merkle tree consistency via $\mathsf{MT.Verify}$. Check if $2\hat{f}_i(\beta^{2^i}) = \hat{f}_{i-1}(\beta^{2^{i-1}}) + \hat{f}_{i-1}(-\beta^{2^{i-1}}) + x_i/\beta^{2^{i-1}} \cdot (\hat{f}_{i-1}(\beta^{2^{i-1}}) - \hat{f}_{i-1}(-\beta^{2^{i-1}}))$, $\forall i \in [\sigma+1]$.*
  5. ***Validity check:** $\mathcal{P}$ and $\mathcal{V}$ invoke rolling batch FRI to prove $\forall i \in [\sigma+1]$, $\hat{f}_{i-1}|_{L_{i-1}} \in \mathrm{RS}[L_{i-1}, 2N/|L_{i-1}|]$, where $\hat{f}_0$ is proved via the batch FRI of $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ and $\hat{s}$. The FRI needs $q' = O(\lambda)$ queries over $\{L_{i-1}\}$, which can be reused from the consistency check.*
  6. *Iff all checks pass and $\hat{f}_{\sigma+1} = y + \alpha y_s$, $\mathcal{V}$ outputs 1.*

Commit. The commitment generation includes $O(\ell)$ FFTs for $O(m)$-length vectors, taking $O(\ell m \log m) = O(N\log m)\mathbb{F}$. These FFTs are independent and parallelizable. In addition, the prover constructs a Merkle tree with $O(m)$ leaves, each containing ℓ entries, which takes $O(m)\mathsf{H}_\ell + O(m)\mathsf{H}$.

Open. In the i-th round, $\mathcal{P}$ computes $\hat{f}_i|_{L_i}$ and Merkle tree commitments from $\hat{f}_{i-1}|_{L_{i-1}}$ in $O(|L_{i-1}|)$ time. The total time complexity is $O(\sum_{i\in[0,\mu]} m/2^i) = O(m)\mathbb{F}/\mathsf{H}$. $\mathcal{P}$ also runs a rolling batch FRI over $\hat{f}_1,\ldots,\hat{f}_\ell$, taking $O(\sum_{i\in[0,\mu]} m/2^i) = O(m)$ field and hash operations. Besides these, the combination of $\hat{f}_0$ from $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ takes a one-time and time-$O(N)$ random linear combination over field, *i.e.*, $O(N)\mathbb{F}_{\mathsf{rlc}}$.

Proof size and verify. In the i-th round, $\mathcal{V}$ opens two trees with $O(\lambda)$ queries, so the total proof size of $\log m$ rounds is $O(\lambda \log m)\,\mathbb{F}$ and $O(\log^2 m)\,\mathsf{H}$. In the first round, $\mathcal{V}$ opens $O(\lambda)$ leaves on an optimized Merkle tree with $O(m)$ leaves (each with ℓ elements), adding $O(\lambda\ell)\,\mathbb{F}$ and $O(\lambda)\,\mathsf{H}_\ell$. The total proof size hence follows. The verifier complexity is similar since it processes the received messages linearly.

Security Analysis. Appendix D provides proof for completeness, binding, knowledge soundness, and zero knowledge.

5 DePip$_{\text{FRI}}$: Distributed FRI-based PCS

We introduce DePip$_{\text{FRI}}$, an ideal distributed FRI-based PCS with generality, accountability, and efficiency. Section 5.1 formalizes and improves a distributed FRI with reduced communication and computation. Section 5.2 builds DePip$_{\text{FRI}}$ using the distributed FRI. Section 5.3 achieves accountability.

5.1 Improved Distributed FRI

We formalize and improve the distributed FRI implicit in [54]. Let $\mathcal{P}_0$ be the master prover, with ℓ sub-provers $\mathcal{P}_j$ assigned $f^{(j)}$. Given a coset L and code rate ρ, distributed FRI proves $f^{(j)}|_L \in \mathrm{RS}[L,\rho]$ for all $j \in [\ell]$. The key idea is to consider a batch FRI for $\sum_{j\in[\ell]} \alpha^{j-1} \cdot f^{(j)}(X)$. Distributed FRI requires distributed commitment and opening for these ℓ polynomials. Below, i, j, or k, and h denote indices of rounds, sub-prover, and elements in the codewords of each round.

Distributed commitment generates the Merkle tree commitment C to $f^{(1)}|_L, \ldots, f^{(\ell)}|_L$. Suppose an integer $d = |L|/\ell$. Each sub-prover $\mathcal{P}_j$ sends its assigned segment of d elements $f^{(j)}|_L[(k-1)d+1], \ldots, f^{(j)}|_L[kd]$ to $\mathcal{P}_k$. $\mathcal{P}_k$ then builds a Merkle tree commitment $\mathsf{H}^{(k)}$ for them.

In DeVirgo, each $\mathsf{H}^{(k)}$ is a direct Merkle root to $d\ell$ leaves

$$\{f^{(j)}|_L[(k-1)d+1], \ldots, f^{(j)}|_L[kd]\}_{j\in[\ell]}.$$

Thus, $\mathcal{P}_k$ holds the k-th part of codewords $f^{(1)}|_L, \ldots, f^{(\ell)}|_L$.

We use the optimized Merkle tree method in Section 4.1 to reduce the tree size. Fixing a k, for any $h \in [(k-1)d+1, kd]$, the ℓ entries $\{f^{(j)}|_L[h]\}_{j\in[\ell]}$ would either be queried together, or none would be queried at all. We then combine the ℓ entries $\{f^{(j)}|_L[h]\}_{j\in[\ell]}$ into a single leaf. $\mathcal{P}_k$ then builds a Merkle tree root $\mathsf{H}^{(k)}$ with d instead of $d\ell$ leaves. Next, $\mathcal{P}_k$ sends $\mathsf{H}^{(k)}$ to $\mathcal{P}_0$, who builds the final Merkle tree root from $\mathsf{H}^{(1)}, \ldots, \mathsf{H}^{(\ell)}$, which reduces to $d\ell = m$ leaves as opposed to $d\ell^2 = m\ell$.

Distributed openings require provers to compute the i-th round polynomial $f_i(X) = g_{i-1}(X) + \alpha_i h_{i-1}(X)$ for $i \in [1, \log m]$. Setting $f_0(X) = \sum_{j\in[\ell]} \alpha^{j-1} \cdot f^{(j)}(X)$, these polynomials satisfy $f_{i-1}(X) = g_{i-1}(X^2) + X \cdot h_{i-1}(X^2)$, where the odd-even decomposition is unique, and α_i is the i-th round challenge. In the first round, $f_0(X)$ can be computed by the distributed commitment above. DeVirgo does not describe the remaining rounds but assumes the same method can be applied. However, applying the first-round distributed commitment in later rounds is less efficient, as detailed below.

A less efficient approach. Like the commitment, sub-provers compute sub-codewords locally, exchange entries, and generate commitments. Let α_i be the shared challenge, $f_i^{(j)}(X) = g_{i-1}^{(j)}(X) + \alpha_i \cdot h_{i-1}^{(j)}(X)$ be the sub-polynomial held by $\mathcal{P}_j$ in round i, and the target polynomial be $f_i(X) = g_{i-1}(X) + \alpha_i \cdot h_{i-1}(X)$. By the property of unique decomposition, we have

$$g_{i-1}(X) = \sum_{j\in[\ell]} g_{i-1}^{(j)}(X), \quad h_{i-1}(X) = \sum_{j\in[\ell]} h_{i-1}^{(j)}(X),$$

and $f_i(X) = \sum_{j\in[\ell]} f_i^{(j)}(X)$. Hence, $\mathcal{P}_j$ can compute $f_i^{(j)}(X)$ locally by folding $f^{(j)}(X)$. Then, similar to the distributed commitment, assuming the i-th round coset to be L_i and $d_i = |L_i|/\ell$, $\mathcal{P}_j$ sends $f_i^{(j)}|_{L_i}[(k-1)d_i+1], \ldots, f_i^{(j)}|_{L_i}[kd_i]$ to $\mathcal{P}_k$, who computes $f_i|_{L_i}[h] = \sum_{j\in[\ell]} f_i^{(j)}|_{L_i}[h]$ for $h \in [(k-1)d_i + 1, kd_i]$ and builds a commitment $\mathsf{H}_i^{(j)}$. Then, sub-provers can compute and commit the i-th round polynomial $f_i(X)$.

We analyze the complexity below. In the i-th round ($i > 1$), $\mathcal{P}_j$ needs to compute $|L_i|$ entries to build $f_i^{(j)}|_{L_i}$ locally, and compute $d_i = |L_i|/\ell$ entries on $f_i|_{L_i}$, taking $O(|L_i| + |L_i|/\ell)$ time. $\mathcal{P}_j$ also receives $(\ell-1)\cdot|L_i|/\ell$ entries from other sub-provers, costing a total communication overhead of $O(|L_i|)$.

Our distributed opening. Our key insight is that in the i-th round, $\mathcal{P}_j$ at least needs $|L_i|/\ell$ entries to build the distributed Merkle tree commitment, *i.e.*, $f_i|_{L_i}[(j-1)d_i+1], \ldots, f_i|_{L_i}[jd_i]$. In the less efficient approach, $\mathcal{P}_j$ computes these d_i entries via $d_i \cdot \ell$ entries, *i.e.*, $\{f_i^{(j)}|_{L_i}[(j-1)d_i+1], \ldots, f_i^{(j)}|_{L_i}[jd_i]\}$ for all $j \in [\ell]$. This is hence not optimal.

We build an optimal distributed method. By the properties of FRI, for any $a \in [|L_i|]$, there exists a function F such that

$$f_i|_{L_i}[a] = F(f_{i-1}|_{L_{i-1}}[a], f_{i-1}|_{L_{i-1}}[a + |L_{i-1}|/2], \alpha_i).$$

Hence, as long as the whole $f_{i-1}|_{L_{i-1}}$ is held by all provers, $\mathcal{P}_j$ only needs to receive $d_i \cdot 2$ entries to compute $f_i|_{L_i}[(j-1)d_i+1], \ldots, f_i|_{L_i}[jd_i]$, which may be sent from different provers.

We next show that for all i, the whole $f_{i-1}|_{L_{i-1}}$ is indeed held by all provers. It suffices to show the case when $i = 1$, other cases are recursive and similar. Recall that in the distributed commitment, $\mathcal{P}_j$ has $\{f_0^{(k)}|_{L_0}[(j-1)d_0+1], \ldots, f_0^{(k)}|_{L_0}[jd_0]\}_{k\in[\ell]}$. and can compute $f_0|_{L_0}[h] = \sum_{k\in[\ell]} \alpha^k \cdot f_0^{(k)}|_{L_0}[h]$ for all $h \in [(k-1)d_0+1, kd_0]$. Hence, the whole $f_0|_{L_0}$ is averagely held by all provers.

For complexity, in round i, $\mathcal{P}_j$ computes only $2d_i = 2 \cdot |L_i|/\ell$ entries in $O(|L_i|/\ell)$ time instead of $O(|L_i| + |L_i|/\ell)$. $\mathcal{P}_j$ receives at most $2 \cdot |L_i|/\ell$ entries from other sub-provers, costing an overhead of $O(|L_i|/\ell)$ instead of the $O(|L_i|)$.

Figure 1 illustrates an example in the first round of distributed evaluation. To compute and distributedly commit

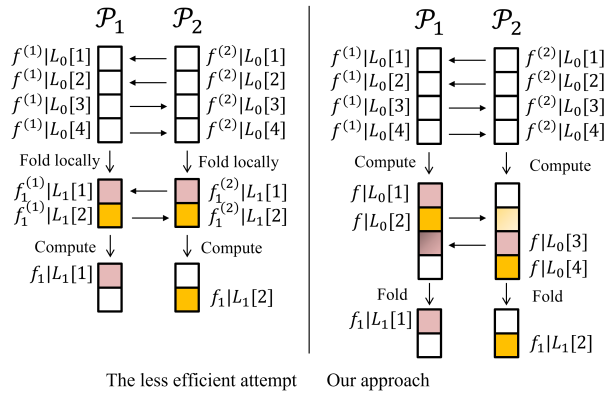

Figure 1: Comparison of two approaches in the first round of distributed evaluation for $\ell = 2$ and $|L_0| = 4$

$f_1|_{L_1}$, the less-efficient method requires computing 2 entries for folding and receives $|L_1|/\ell \cdot (\ell - 1)$ entries per sub-prover, while our approach requires only 1 and receives at most $|L_1|/\ell \cdot 2$ entries per sub-prover. Here, "at most" means that a sub-prover may already have the needed entries (*e.g.*, $\mathcal{P}_1$ holds $f^{(1)}|_{L_0}[1]$ and $f^{(2)}|_{L_0}[1]$, and can compute $f|_{L_0}[1] = \alpha f^{(1)}|_{L_0}[1] + \alpha^2 f^{(2)}|_{L_0}[1]$). For $\ell \geq 4$, our method further reduces the communication overhead.

Based on the distributed commitment and our improved distributed opening, Protocol 5 in Appendix E formally presents distributed FRI. It mainly includes four procedures: 1) Step 1, the distributed commitment; 2) Step 3, the distributed opening for first $\log(m/\ell)$ rounds; 3) Step 4, the opening for last $\log \ell$ rounds; 4) Step 5, the verification. We argue that after $\log(m/\ell)$ rounds, the codewords held by provers are of dimension ℓ. Then, sub-provers can send their codewords directly to the master prover, who runs the following protocol non-distributedly. The time complexity is $O(\ell)$.

Theorem 5.1. *Protocol 5 is a distributed FRI. In the committing phase, the sub-prover complexity is $O(m \log m)$, the amortized communication complexity is $O(m)$, and the master prover complexity is $O(\ell)$. In the opening phase, the sub-prover complexity is $O(m/\ell)$, the amortized communication complexity is $O(m/\ell)$, and the master prover complexity is $O(\ell \log m)$. The proof size and verifier complexity are $O(\lambda \log m + \lambda\ell + \log^2 m)$, or $O(\lambda \log m + \lambda\ell)\,\mathbb{F} + O(\log^2 m)\,\mathsf{H} + O(\lambda)\,\mathsf{H}_\ell$ in detail.*

The detailed proofs are presented in Appendix E.

5.2 Construction of DePIP$_{\text{FRI}}$

Equipped with our distributed FRI, this section presents DePIP$_{\text{FRI}}$. For generality, we use the shred-to-shine method in PIP$_{\text{FRI}}$. Recall that given a size-N multilinear polynomial $\tilde{f}$, PIP$_{\text{FRI}}$ splits it into ℓ size-m multilinear ones $\tilde{f}^{(1)}, \ldots, \tilde{f}^{(\ell)}$. The FFT of $\hat{f}$ is then transformed into FFTs of $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$. This can be executed by provers locally, thereby avoiding a single linear-size FFT that is hard to distribute.

Distributed committing. The inputs of DePIP$_{\text{FRI}}$ are the same as those of the distributed FRI. Hence, DePIP$_{\text{FRI}}$'s committing algorithm can directly use Step 1 of Protocol 5. We assume each of ℓ sub-provers holds one polynomial here. Extending one prover to hold more polynomials is natural.

Distributed opening includes two FRI-like procedures. Step 3 of PIP$_{\text{FRI}}$ follows the FRI method with folding parameters set as evaluation point variables rather than random challenges. Sub-provers mimic distributed committing and opening of the distributed FRI (Steps 3 and 4) to generate, commit, and open $\hat{f}_1, \ldots, \hat{f}_\sigma$, where $\sigma = \log m$. Thereafter, the prover and verifier execute a batch FRI over $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$ and a rolling batch FRI over $\hat{f}_1, \ldots, \hat{f}_\sigma$. We now explain how to adapt the distributed FRI over $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$ into a distributed rolling batch FRI over both $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$ and $\hat{f}_1, \ldots, \hat{f}_\sigma$.

After generating $\hat{f}_1|_{L_1}, \ldots, \hat{f}_\sigma|_{L_\sigma}$ of the distributed "FRI," sub-prover $\mathcal{P}_j$ holds the j-th part of $\hat{f}_i|_{L_i}$ of length $d_i = |L_i|/\ell$, where $i \in [\sigma]$. By the distributed FRI over $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$, in round i, $\mathcal{P}_j$ also obtains the j-th part of $f_i|_{L_i}$, *the i-th folded codeword of the batch FRI* for $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$. To run rolling batch FRI, the sub-provers compute $f'_{i+1}|_{L_{i+1}}$ such that

$$\begin{aligned} f'_{i+1}(X) &= g_i(X) + \alpha_i \cdot h_i(X) + \alpha_i^2 \cdot \hat{f}_{i+1}(X) \\ &= f_{i+1}(X) + \alpha_i^2 \cdot \hat{f}_{i+1}(X), \end{aligned} \tag{2}$$

where $f_i(X)$ is uniquely decomposed as $g_i(X^2) + X \cdot h_i(X^2)$ and $f_{i+1}(X) = g_i(X) + \alpha_i \cdot h_i(X)$. By the property of distributed FRI, $\mathcal{P}_j$ can compute the j-th part of $f_{i+1}|_{L_{i+1}}$, and hence j-th part of $f'_{i+1}|_{L_{i+1}}$ as well. This additional computation requires only $O(|L_i|/\ell)$ work per round over distributed FRI, leaving the overall complexity unchanged.

With the distributed rolling batch FRI, we build DePIP$_{\text{FRI}}$ in Protocol 6, Appendix F. For simplicity, we give a non-zk version; extending it to zero knowledge is natural.

Theorem 5.2. *Protocol 6 is a distributed MLPCS. The complexities are the same as Protocol 5 in Theorem 5.1.*

Security and Complexity. DePIP$_{\text{FRI}}$ is equivalent to PIP$_{\text{FRI}}$ from the verifier's view, assuming each sub-prover is honest. Security properties hence follow from Theorem 4.1. DePIP$_{\text{FRI}}$ runs the distributed FRI twice, with one instance modified into a distributed rolling batch FRI without affecting complexities. Hence, the proof size and verifier complexity are $O(\lambda \log m + \lambda\ell + \log^2 m)$, the same as PIP$_{\text{FRI}}$. With $\ell \leq \log^2 N$, they are smaller than $O(\log^2 N)$, the same as existing FRI-based MLPCSs such as PolyFRIM and DeepFold. It is feasible since ℓ, the prover number, is usually small compared to N, potentially as large as 2^{30} in distributed proofs.

5.3 Achieving Accountability

Section 5.2 assumes honest sub-provers; we now add accountability so an honest master prover can flag sub-provers who submit wrong proofs despite holding the true witness.

We first explain the challenges. In the verification phase, the verifier checks: 1) the consistency of Merkle tree paths and queried entries; 2) the consistency of multiple triples of queried entries, *e.g.*, if $f_i(\beta^2) = F(f_{i-1}(-\beta), f_{i-1}(\beta), \alpha_i)$. A clever malicious sub-prover would not violate 1), as her sub Merkle tree is generated and opened independently. Violating this would be directly detected. For 2), even if the master prover detects inconsistency, as $f_i(\beta^2)$, $f_{i-1}(-\beta)$ and $f_{i-1}(\beta)$ come from different sub-provers, say, $\mathcal{P}_{j_1}, \mathcal{P}_{j_2}$ and $\mathcal{P}_{j_3}$, the master prover cannot pinpoint the exact malicious sub-prover. Even dropping all $\mathcal{P}_{j_1}, \mathcal{P}_{j_2}$ and $\mathcal{P}_{j_3}$ does not work, as the malice may occur in the earlier rounds, *e.g.*, $f_{i-1}(\beta)$ is computed honestly but the earlier $f_{i-2}(\beta^{1/2})$ or $f_{i-2}(-\beta^{1/2})$ is wrong.

To address the challenges, we let sub-provers additionally generate sub-proofs for their sub-polynomials. By soundness and the construction of PIP$_{\text{FRI}}$, these sub-proofs help construct authentic reference proof. By comparing these sub-proofs with the distributed proofs, the master prover can precisely identify the malicious sub-prover.

The concrete modifications are described as follows.

1. For $i \in [\ell]$, $\mathcal{P}_i$ invokes PIP$_{\text{FRI}}$ to additionally generate and send to $\mathcal{P}_0$ the sub-commitments for $f^{(i)}(X)$. $\mathcal{P}_i$ also sends to $\mathcal{P}_0$ the evaluation $f^{(i)}(\boldsymbol{x'})$. $\mathcal{P}_0$ sets the claimed evaluation as $f(\boldsymbol{x}) = \sum_{i\in[\ell]} w_i \cdot f^{(i)}(\boldsymbol{x'})$. The completeness holds by Equation (1) and Protocol 1.
2. Assume in the j-th round, $\mathcal{P}_0$ firstly detects inconsistent entry triples, say, $f_{j-1}(\beta^{2^{j-1}}), f_{j-1}(-\beta^{2^{j-1}}), f_j(\beta^{2^j})$ for some query β. We note that by the construction of DEPIP$_{\text{FRI}}$, it holds that $f_j(X) = \sum_{i\in[\ell]} \alpha_i \cdot f_j^{(i)}(X)$, where $f_j^{(i)}(X)$ means the j-th round folded polynomial of $f^{(i)}(X)$.
3. Given the queries β from the verifier, $\mathcal{P}_0$ acts as a verifier to ask and check all sub-proofs for $\{f^{(i)}(\boldsymbol{x'})\}_{i\in[\ell]}$. As the sub-proofs are generated independently, if some sub-proof fails, $\mathcal{P}_0$ detects a certain malicious sub-prover. If all sub-proofs pass, by the soundness of PIP$_{\text{FRI}}$, with a high probability $\mathcal{P}_0$ has the **true** tuples $\{f_{j-1}^{(i)}(\beta^{2^{j-1}})\}_{i\in[\ell]}$, $\{f_{j-1}^{(i)}(-\beta^{2^{j-1}})\}_{i\in[\ell]}$, and $\{f_j^{(i)}(\beta^{2^j})\}_{i\in[\ell]}$.
4. W.l.o.g., assume $f_{j-1}(\beta^{2^{j-1}})$ is incorrect. Then, $\mathcal{P}_0$ can compute the **true** $f'_{j-1}(\beta^{2^{j-1}}) = \sum_{i\in[\ell]} \alpha_i \cdot f_{j-1}^{(i)}(\beta^{2^{j-1}})$ and detects inconsistency. $\mathcal{P}_0$ then traces the joint proof before $f_{j-1}(\beta^{2^{j-1}})$ and finds all inconsistencies between sub-proofs and joint proofs, revealing all malicious sub-provers.

Adding accountability to DEPIP$_{\text{FRI}}$ requires each sub-prover to generate a sub-proof for its size-m sub-polynomial at cost $O(m \log m)$, the same as in the original protocol. The master prover then verifies ℓ sub-proofs, raising its cost from $O(\ell \log m)$ to $O(\ell \log^2 m)$; which remains far below a sub-prover's workload as $m \gg \ell$. Communication grows by $O(\ell \log^2 m)$, still much smaller than the original $O(m\ell)$. We thus achieve accountability with no asymptotic overhead, offering a blueprint for other FRI-based distributed PCSs.

6 Implementation and Evaluation

We implemented PIP$_{\text{FRI}}$ and DEPIP$_{\text{FRI}}$ in Rust using the arkworks ecosystem.[3] We ran non-distributed experiments on an AMD Ryzen 3900X processor with 12 cores and 32 GB RAM, and ran distributed experiments on an Intel Xeon Platinum 8255C processor with 24 cores. For distributed experiments, each sub-prover is a core, and we use 2-16 cores. Reported figures are averages over 10 executions. Unless explicitly stated, all experiments assume no parallelization.

6.1 Evaluation of PCSs

Choices of Comparisons. We compare with several state-of-the-art code-based MLPCSs including Virgo, Orion [25], PolyFRIM [26], and DeepFold [21]. We did not benchmark several schemes below for the following reasons.

- HyperPlonk [19] and Basefold [58] perform worse than DeepFold, and we use DeepFold in our benchmarks.
- Brakedown [32] and Block *et al.* [11] are field-agnostic MLPCSs with linear prover complexity. They are reportedly or estimably worse than the non-field-agnostic Orion on RS-friendly fields. Our benchmarks run on RS-friendly fields.
- Blaze [14] runs over binary fields. We run over prime ones.

Apart from code-based PCSs, we also compare group-based mKZG [23] and Hyrax [4]. The former has $O(\log N)$ proof size and verification but needs trusted setups. The latter is transparent, with $O(\sqrt{N})$ proof size and verification.

Parameters. Our field is $\mathbb{F}_p$ with $p = 2^{64} - 2^{32} + 1$, also known as the Goldilocks field, as used in plonky2 [46].[4] The number of sub-polynomials is set as the nearest power-of-two integer of $\log_2(4 \times \mu)$; in our experiments, this value is 64 or 128. For most schemes, we set the RS code rate as $\rho = 1/8$, the same as PolyFRIM and DeepFold. We also implement a PIP$_{\text{FRI}}$ variant with $\rho = 1/2$, trading proof size and verification time for prover time. Our security level is $\lambda = 100$ bits. In FRI-based schemes, the query repetition number (q' in Protocol 2) is $-\lambda / \log_2 \rho$, providing a conjectured security [8]. This conjecture underlies both academic [34, 56, 59, 60] and industrial works [46, 48, 54], ensures list polynomial binding, *i.e.*, the committed vector binds to several polynomials, and DEEP-FRI [9] can reduce them into one with slight

[3] We will open-source our implementations soon.

[4] All our benchmarks, except Orion in C++, use arkworks in Rust. All the code-based benchmarks use $\mathbb{F}_p$, except Orion uses $\mathbb{F}_{q^2}$ where $q = 2^{61} - 1$. Basic operations over $\mathbb{F}_{q^2}$ like FFTs can be 70% slower than $\mathbb{F}_p$. However, the advantages of PIP$_{\text{FRI-1/2}}$ still remain after counting these gaps.

overhead (*cf.*, DeepFold vs. Basefold). For the direct LDT used in Orion, we (re-)set the queried column number as $-\lambda/\log_2(1-d/3)$ [28, 32], where d is the GS code distance.

Performance of PIP_{FRI}. Figure 2 compares *non-zk* MLPCSs for polynomial sizes from 2^{18} to 2^{27}. The prover time can be obtained by adding the committing and opening times. Any absent data is due to out-of-memory issues, except for Orion. (We failed to run Orion when the polynomial size exceeded 2^{23}.) All schemes run normally for $\mu = 22$. PIP_{FRI} and Hyrax are the only PCSs supporting $\mu = 27$. For such multilinear polynomials, $\text{PIP}_{\text{FRI-1/8}}$ features a 115s prover time, a 2.8ms verifier time, and a 358KB proof size.

Compared with mKZG, $\text{PIP}_{\text{FRI-1/8}}$ is 10-15× faster in prover and 5× faster in verifier despite the worse complexity. Its proof size can be 100× larger. However, mKZG needs a trusted setup. Compared with Hyrax, $\text{PIP}_{\text{FRI-1/8}}$ is 10× faster in prover time. $\text{PIP}_{\text{FRI-1/8}}$'s verifier has 30-300× faster due to its lower complexity. For small μ, Merkle tree openings cause $\text{PIP}_{\text{FRI-1/8}}$ to yield larger proofs, but for $\mu \geq 27$, its proof size becomes smaller due to lower complexity.

Compared with Virgo and PolyFRIM, $\text{PIP}_{\text{FRI-1/8}}$ is 20× faster in prover, 1.1-1.6× faster in verifier, and 1.4-1.8× smaller in proof size. Compared with DeepFold, $\text{PIP}_{\text{FRI-1/8}}$ is 5× faster in committing and 40× faster in opening, with a 14× faster prover time. Its proof size and verifier time are competitive, ranging from 5% better to 7% worse. Compared with Orion with linear prover complexity, $\text{PIP}_{\text{FRI-1/8}}$ has only 5% slower prover time, but is 30× better proof size and verifier time. By setting the code rate to $1/2$, $\text{PIP}_{\text{FRI-1/2}}$ has even a 3.5× faster prover time, with 15× proof-size and verification advantages. As far as we know, this is the first FRI-based PCS achieving competitive or faster prover than Orion.

Figure 2(e) compares memory costs, which are crucial for the scalability of SNARKs, especially in large-scale tasks. For example, in zkLLM [49], a zero-knowledge proof for large language models, there can be 13 billion parameters, *i.e.*, requiring MLPCSs with $\mu \approx 34$. In memory use, $\text{PIP}_{\text{FRI-1/8}}$ matches Orion, which is 4× less than mKZG, and at least 10× less than Virgo, PolyFRIM, and DeepFold. $\text{PIP}_{\text{FRI-1/2}}$ needs 3× less memory than $\text{PIP}_{\text{FRI-1/8}}$ due to shorter RS codewords, and is 30% smaller than Hyrax. $\text{PIP}_{\text{FRI-1/2}}$ is the most memory-efficient in Figure 2 and is the sole PCS supporting polynomials with $\mu = 29$ given a 32 GB RAM. This makes PIP_{FRI} promising for large-scale computations. Appendix G.1 gives the concrete data of Figure 2.

Further comparisons with WHIR. Recent work WHIR [6] proposes a new LDT for RS codes with faster verifier complexity, fewer queries, and hence smaller proof size than FRI. WHIR is an MLPCS constructed by the LDT and the multilinear sum-check in Basefold and DeepFold. As shown in Table 7, PIP_{FRI} has a 5× faster prover time than WHIR, but with 1.8× slower verifier time and 2× larger proof size. However, this comparison has two factors in WHIR's favor.

First, WHIR is a drop-in replacement of FRI. The advantages of WHIR's proof size and verifier time are due to the better complexities of WHIR's LDT over FRI [6]. Similar to PIP_{FRI} combining FRI and our *general* framework PIP, we can build an MLPCS combining WHIR's LDT and PIP. This would at least reduce the gaps in proof size and verifier time.

Second, the FRI benchmark in WHIR has a 2× faster prover and 1.2× faster verifier than our FRI benchmark. This is probably due to the different implementations of Merkle trees. Using WHIR's FRI can make PIP_{FRI} even faster.

Overall, these factors combine to indicate that PIP_{FRI} can achieve better time performance in practice.

Table 7: Comparisons of WHIR and PIP_{FRI}

Scheme	Size	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Size	$\mathcal{P}$	$\mathcal{V}$	π
WHIR	2^{20}	3.8	0.9	85	2^{22}	17.7	1.0	99
PIP_{FRI}		0.8	1.5	156		3.3	1.8	196

Table 8: Performance of code-based zk-PCSs

Scheme	Size	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Size	$\mathcal{P}$	$\mathcal{V}$	π
FRI-PCS	2^{18}	4.8	2.3	242	2^{20}	22.1	2.9	306
Virgo		23.2	1.8	212		102.3	2.1	265
DeepFold		5.1	1.3	137		22.5	1.6	169
PIP_{FRI}		0.4	1.3	132		1.8	1.6	175

Performance of zk-PCSs. As presented in Table 8, zk-PIP_{FRI} is 10-50× faster than FRI-PCS and Virgo in prover time, with 1.3-1.8× faster verifier time and 1.5-1.8× smaller proof size. Compared with zk-DeepFold, zk-PIP_{FRI} is 10× faster in prover time, with competitive proof size and verifier time.

Notably, (zk-)PIP_{FRI} outperforms (zk-)FRI-PCS in all aspects. This seems "impossible" as all FRI-based PCSs use FRI with at least the *same total instance size* as sub-protocols. We achieve this by the shred-to-shine framework PIP to make *more* but *smaller* FRI instances and combine them efficiently.

6.2 Evaluation of SNARKs

Figure 3 shows the performance of SNARKs using PCSs and PIOPs in Spartan [47] or HyperPlonk [19], which are state-of-the-art linear-prover PIOPs for R1CS and Plonkish. We use open-sourced benchmarks of the HyperPlonk PIOP [34] and Spartan [27]. Spartan offers two PIOPs; we use the one with linear verification, as the sublinear one depends on a specific group-based PCS, Hyrax. Hence, schemes in Figure 3 use the Spartan PIOP feature linear verifier time.

PIP_{FRI} + HyperPlonk offers a 100s prover time for a circuit with 2^{23} constraints. It is 2× faster than DeepFold with a 3% worse proof size and verifier time. Compared to Orion, PIP_{FRI} + HyperPlonk has a 2% worse prover time but is 10-20× better in proof size and verifier time. When combined with Spartan, PIP_{FRI} results in a SNARK with a 10s prover

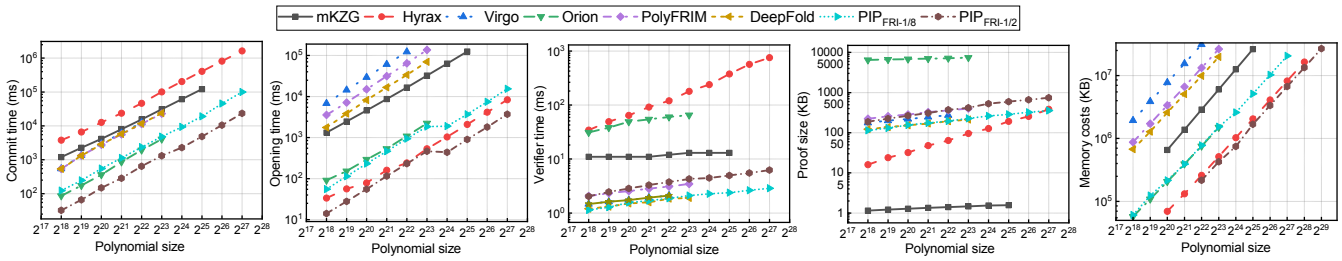

Figure 2: Performance comparison of PIP_{FRI}, mKZG, Hyrax, and several other code-based MLPCSs

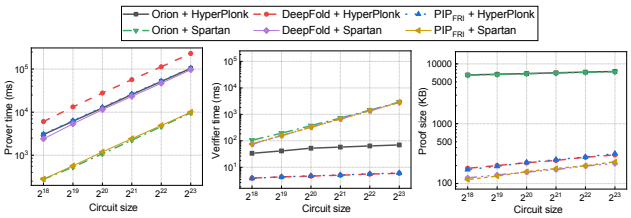

Figure 3: SNARKs from code-based MLPCSs and PIOPs in Spartan or HyperPlonk

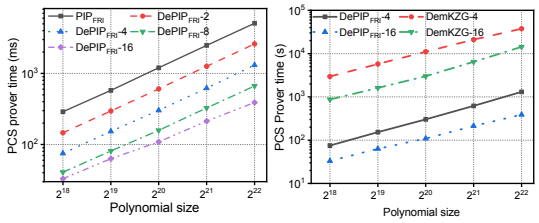

Figure 4: (a) Prover times of PIP_{FRI} and $\text{DEPIP}_{\text{FRI}}$; (b) Prover times of $\text{DEPIP}_{\text{FRI}}$ and DemKZG, with different sub-provers

for a size-2^{23} circuit, which is 10× better than DeepFold, with close proof size and verifier time. Compared with Orion, PIP_{FRI} + Spartan shows a 20× reduction in proof size, with close prover time and a slightly better verifier time. These experiments show that PIP_{FRI} enhances existing SNARKs.

6.3 Evaluation of Distributed Protocols

Figure 4 (a) shows the prover (commit + open) time of $\text{DEPIP}_{\text{FRI}}$ with 2-16 provers. $\text{DEPIP}_{\text{FRI}}$ achieves full linear speedup. For a size-2^{22} polynomial, its prove time is 0.66s with 8 provers, 7.6× faster than PIP_{FRI}. The proof size and verifier time are basically the same as PIP_{FRI}.

Figure 4 (b) and Table 9 compare $\text{DEPIP}_{\text{FRI}}$ and other distributed PCSs, including DemKZG [40] (with 4 or 16 provers) and DeVirgo.[5] We omit to compare with the transparent DeDory [40], which is less efficient than DemKZG in all aspects.

[5]We failed to obtain open-sourced implementations of DeVirgo, and estimate its performance by Virgo, assuming its *full linear speedup*. We also estimate the communication via our implemented distributed Merkle trees.

Table 9: Performance of distributed PCSs for size-2^{22} polynomials with 16 provers

Scheme	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Per comm. (KB)
DemKZG	14.3	17	2.3	1.7
DeVirgo	7.7	2.2	213	110,000
$\text{DEPIP}_{\text{FRI}}$	0.39	2.1	198	15,000

Table 10: Performance of distributed SNARKs for size-2^{22} circuits with 16 provers, built by different distributed PCSs

PCS	Circuit	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Per comm. (KB)
DemKZG	Arbitrary	18.2	17.4	10.7	16
DeVirgo	Parallel	-	-	-	125,000
$\text{DEPIP}_{\text{FRI}}$	Arbitrary	4.3	2.5	206.4	46,000

As shown, equipped with the same provers, the prover time of $\text{DEPIP}_{\text{FRI}}$ is 40× faster than demKZG, and 20× faster than DeVirgo. Notably, $\text{DEPIP}_{\text{FRI}}$ also reduces the amortized communication overhead of DeVirgo by 7× while maintaining competitive proof size and verifier time.

HyperPianist [40] is the state-of-the-art distributed SNARK with efficient prover constructed by the distributed PIOP variant of HyperPlonk [19] and the distributed demKZG PCS. Table 10 shows the performance of distributed SNARKs built from different distributed PCSs. $\text{DEPIP}_{\text{FRI}}$ leads to a distributed SNARK with a 4× faster prover and 7× faster verifier than the original HyperPianist with demKZG. $\text{DEPIP}_{\text{FRI}}$ further leads to a *general* SNARK with 3× smaller communication than the *non-general* scheme from DeVirgo. For non-general schemes, our SNARK has an even smaller communication as the general scheme requires three distributed PCSs [40] while the non-general scheme only requires one.

6.4 Applications to Machine Learning

PIP_{FRI} has efficient prover times and low memory costs, making it well-suited for large-scale computations. Kaizen [2], a zero-knowledge proof of training, uses code-based PCS Virgo as sub-protocols, where the PCS committing time reportedly exceeds 50%. Using PIP_{FRI} reduces prover time by 36%-

42%, as shown in Table 12 in Appendix G.3. For instance, for 4 batches of LeNet ($N = 4$), PIP_{FRI} cuts the PCS time from 27.8 seconds with Virgo to just 5.7 seconds, reducing the PCS's share of the total time from 54.2% to only 19.5%.

7 Conclusion and Future Work

We propose a prover-efficient MLPCS PIP_{FRI}, which benefits state-of-the-art prover-efficient (zk-)SNARKs from the "PIOP + MLPCS" paradigm. We also build a code-based distributed MLPCS with accountability and generality, which yields the first code-based distributed SNARK for arbitrary circuits. Experiments show their efficiency, especially in prover times and communication. Further works may explore applying our framework to other PCSs.

References

[1] Kaveh Aasaraai, Emanuele Cesena, Rahul Maganti, Nicolas Stalder, Javier Varela, and Kevin Bowers. Cyclone-NTT: An NTT/FFT architecture using quasi-streaming of large datasets on DDR- and HBM-based FPGA platforms. In *ACM/SIGDA FPGA*, 2023.

[2] Kasra Abbaszadeh, Christodoulos Pappas, Dimitrios Papadopoulos, and Jonathan Katz. Zero-knowledge proofs of training for deep neural networks. In *CCS*, 2024.

[3] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, 2017.

[4] arkworks contributors. Polynomial commitments. https://github.com/arkworks-rs/poly-commit, 2022. Last accessed: 18th July 2025.

[5] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed-Solomon proximity testing with fewer queries. In *CRYPTO Part X*, 2024.

[6] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed-Solomon proximity testing with super-fast verification. In *EUROCRYPT Part IV*, 2025.

[7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *ICALP*, 2018.

[8] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed-Solomon codes. In *FOCS*, 2020.

[9] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In *ITCS*, 2020.

[10] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *CCS*, 2020.

[11] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic SNARKs from expand-accumulate codes. In *CRYPTO Part X*, 2024.

[12] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *CRYPTO Part I*, 2021.

[13] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. IACR ePrint 2019/1021.

[14] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast SNARKs from interleaved RAA codes. In *EUROCRYPT Part IV*, 2025.

[15] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *S&P*, 2018.

[16] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *EUROCRYPT Part I*, 2020.

[17] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *ASIACRYPT Part III*, 2021.

[18] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In *ASIACRYPT Part III*, 2021.

[19] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT Part II*, 2023.

[20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT Part I*, 2020.

[21] DeepFold contributors. Benchmarking DeepFold. https://github.com/guo-yanpei/deepfold-bench, 2025. Last accessed: 18th July 2025.

[22] HyperPianist contributors. HyperPianist: Pianist with linear-time prover and logarithmic communication cost. https://github.com/AntCPLab/HyperPianist, 2025. Last accessed: 18th July 2025.

[23] HyperPlonk contributors. HyperPlonk library. https://github.com/EspressoSystems/hyperplonk, 2023. Last accessed: 18th July 2025.

[24] Kaizen contributors. Kaizen. https://github.com/zkPoTs/kaizen, 2024. Last accessed: 18th July 2025.

[25] Orion contributors. Orion: Zero knowledge proof with linear prover time. https://github.com/sunblaze-ucb/Orion, 2025. Last accessed: 18th July 2025.

[26] PolyFRIM contributors. Benchmarking PolyFRIM and FRAVSS. https://github.com/guo-yanpei/PolyFRIM, 2025. Last accessed: 18th July 2025.

[27] Spartan contributors. Spartan: High-speed zkSNARKs without trusted setup. https://github.com/microsoft/Spartan, 2021. Last accessed: 18th July 2025.

[28] Thomas den Hollander and Daniel Slamanig. A crack in the firmament: Restoring soundness of the Orion proof system and more. IACR ePrint 2024/1164.

[29] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO Part II*, 2018.

[30] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PlonK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR ePrint 2019/953.

[31] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[32] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO Part II*, 2023.

[33] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.

[34] Yanpei Guo, Xuanming Liu, Kexi Huang, Wenjie Qu, Tianyang Tao, and Jiaheng Zhang. DeepFold: Efficient multilinear polynomial commitment from Reed-Solomon code and its application to zero-knowledge proofs. In *USENIX Security*, 2025.

[35] Kobi Gurkan, Ariel Gabizon, and Zac Williamson. Cheon's attack and its effect on the security of big trusted setups. https://ethresear.ch/t/cheons-attack-and-its-effect-on-the-security-of-big-trusted-setups. Last accessed: 18th July 2025.

[36] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

[37] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitment. In *CCS*, 2019.

[38] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. *Journal of Cryptology*, 37(4):38, 2024.

[39] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC Part II*, 2021.

[40] Chongrong Li, Yun Li, Pengfei Zhu, Wenjie Qu, and Jiaheng Zhang. Hyperpianist: Pianist with linear-time prover via fully distributed hyperplonk. In *S&P*, 2025.

[41] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *S&P*, 2024.

[42] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *CCS*, 2021.

[43] Weiliang Ma, Qian Xiong, Xuanhua Shi, Xiaosong Ma, Hai Jin, Haozhao Kuang, Mingyu Gao, Ye Zhang, Haichen Shen, and Weifang Hu. GZKP: A GPU accelerated zero-knowledge proof system. In *ASPLOS*, 2023.

[44] Ueli M. Maurer. Abstract models of computation in cryptography. In *IMACC*, 2005.

[45] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, 2013.

[46] Polygon Zero Team. Plonky2: Fast recursive arguments with PLONK and FRI. https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf, 2022.

[47] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO Part III*, 2020.

[48] StarkWare Team. ethSTARK documentation version 1.2. IACR ePrint 2021/582.

[49] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *CCS*, pages 4405–4419, 2024.

[50] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. IACR ePrint 2019/1020.

[51] Riad S. Wahby, Ioanna Tzialla, Abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *S&P*, 2018.

[52] Wenhao Wang, Fangyan Shi, Dani Vilardell, and Fan Zhang. Cirrus: Performant and accountable distributed SNARK. IACR ePrint 2024/1873.

[53] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In *USENIX Security*, 2018.

[54] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In *CCS*, pages 3003–3017, 2022.

[55] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO Part III*, 2019.

[56] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *CRYPTO Part IV*, 2022.

[57] Hua Xu, Mariana Gama, Emad Heydari Beni, and Jiayi Kang. FRIttata: Distributed proof generation of FRI-based SNARKs. IACR ePrint 2025/1285.

[58] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In *CRYPTO Part X*, 2024.

[59] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P*, 2020.

[60] Zongyang Zhang, Weihan Li, Yanpei Guo, Kexin Shi, Sherman S. M. Chow, Ximeng Liu, and Jin Dong. Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing. In *USENIX Security*, 2024.

A Supplementary Preliminaries

A.1 Argument of Knowledge

An interactive argument for an NP relation $\mathcal{R}$ includes a public parameter generation algorithm $\mathcal{G}$ and a pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$. $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ generates the public parameter. $\mathcal{P}$ and $\mathcal{V}$ represent a PPT prover and verifier, respectively. $\mathcal{P}$ tries to convince $\mathcal{V}$ of the existence of $\mathtt{w}$ s.t. $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ for a statement $\mathtt{x}$ through interaction. An argument of knowledge further allows $\mathtt{w}$ to be efficiently extractable by an extractor.

Definition A.1 (Argument of knowledge)**.** *$(\mathcal{G}, \langle \mathcal{P}, \mathcal{V} \rangle)$ is an interactive argument of knowledge (AoK) for $\mathcal{R}$ if it satisfies:*

- ***Completeness.*** *For every* pp *and all* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$,

$$\Pr[\langle \mathcal{P}(\mathtt{w}), \mathcal{V} \rangle(\mathsf{pp}, \mathtt{x}) = 1] = 1.$$

- ***Knowledge Soundness.*** *For any PPT $\mathcal{P}^*$, there exists an expected polynomial time extractor $\mathcal{E}^{\mathcal{P}^*}$ with access to $\mathcal{P}^*$'s*

randomness s.t. $\forall \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ *and* $\mathtt{x}$*, the following probability is* $\mathsf{negl}(\lambda)$*:*

$$\Pr[\langle \mathcal{P}^*(), \mathcal{V}\rangle(\mathsf{pp}, \mathtt{x}) = 1, (\mathtt{x}, \mathtt{w}) \notin \mathcal{R} \,|\, \mathtt{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, \mathtt{x})].$$

$\mathcal{E}^{\mathcal{P}^*}$ *means* $\mathcal{E}$ *has access to the randomness of* $\mathcal{P}^*$*.*

A public-coin interactive AoK can be transformed into non-interactive via the standard Fiat–Shamir transformation. An AoK is succinct with proof size sublinear to $|\mathtt{w}|$*, or* $O(|\mathtt{x}|)$ *for NP problems.*

A.2 Polynomial Commitment Scheme

We recall the security properties of PCSs [32].

- **Completeness.** For any f with $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$, $C \leftarrow \mathsf{Com}(\mathsf{pp}, f)$, and $f(\boldsymbol{x}) = y$, $\Pr[\mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; f) = 1] = 1$.
- **Polynomial Binding.** For all $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$, any PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu), (C, f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, f_0), b_1 \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, f_1) : \\ b_0 = b_1 = 1 \wedge f_0 \neq f_1 \end{array}\right].$$

- **Knowledge Soundness.** Eval is an AoK for the following NP relation $\mathcal{R}_{\mathsf{Eval}}(\mathsf{pp})$ given $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$:

$$\{(C, \boldsymbol{x}, y; f) \colon f \in \mathbb{F}[d^\mu] \wedge f(\boldsymbol{x}) = y \wedge \mathsf{VerPoly}(\mathsf{pp}, C, f) = 1\}.$$

We say a PCS is zero knowledge [59] if it satisfies:

- **Honest-Verifier Zero Knowledge.** For all PPT adversaries $\mathcal{A}$, randomness $r_{\mathcal{A}}$, $\mathsf{pp} \leftarrow \mathsf{Gen}\,(1^\lambda, d, \mu)$, and size-$d^\mu$ polynomial f, there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ s.t. the experiments in Figure 5 are computationally indistinguishable, *i.e.*, $|\Pr[\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp}) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp}) = 1]| \leq \mathsf{negl}(\lambda)$. If the difference is 0, we call it perfect zero knowledge.

$\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp})$:	$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp})$:
1: $C \leftarrow \mathsf{Com}(f, \mathsf{pp})$	1: $C \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{pp}, r_{\mathcal{A}})$
2: $\boldsymbol{x} \leftarrow \mathcal{A}(C, \mathsf{pp})$	2: $\boldsymbol{x} \leftarrow \mathcal{A}(C, \mathsf{pp})$
3: $y \leftarrow f(\boldsymbol{x})$	3: $y \leftarrow f(\boldsymbol{x})$
4: $1 \leftarrow \langle \mathcal{P}(f), \mathcal{A}\rangle(\mathsf{pp}, C, \boldsymbol{x}, y)$	4: $1 \leftarrow \langle \mathcal{S}_2, \mathcal{A}\rangle(\mathsf{pp}, C, \boldsymbol{x}, y)$, given oracle access to y
5: $b \leftarrow \mathcal{A}$ and output b	5: $b \leftarrow \mathcal{A}$ and output b

Figure 5: Experiments for zero knowledge in PCSs

We call the committing complexity of a PCS the time complexity of the Com algorithm. The opening complexity is the time complexity of the prover in the Eval protocol. The verifier complexity is the time complexity of the verifier in the Eval protocol. The proof size is the proof size of the Eval protocol. In a typical SNARK, the prover of SNARK generates the PCS commitments and acts as the prover in the Eval protocol. Hence, we use the prover complexity to denote the larger one in committing complexity and opening complexity.

Protocol 3 (The Rolling Batch FRI [60]). ***Inputs:*** *Secret codewords* $f_0|_{L_0}, \ldots, f_\sigma|_{L_\sigma}$ *such that for* $i \in [0, \sigma]$, $f_i|_{L_i} \in \mathrm{RS}[L_i, \rho]$*. Public multiplicative cosets* $\{L_i\}_{i\in[0,\sigma]}$ *such that* $L_{i+1} = \{x^2 | x \in L_i\}$ *for* $i \in [0, \sigma]$*. Without loss of generality, assume the degree of* f_σ *is* 1*.*
Prover $\mathcal{P}$ *and verifier* $\mathcal{V}$ *run the following algorithms to prove* $f_i|_{L_i} \in \mathrm{RS}[L_i, \rho]$ *for all* $i \in [0, \sigma]$*:*

1. $\mathcal{P} \to \mathcal{V}$*:* $\{C_i\}_{i\in[0,\sigma]}$ *such that* $C_i \leftarrow \mathsf{MT.Commit}(f_i|_{L_i})$.
2. *For* $i \in [0, \sigma]$*,* $\mathcal{P}$ *and* $\mathcal{V}$ *do:*
   (a) $\mathcal{P}$*: sets* $f'_0 = f_0$*. Decompose* $f'_i(X)$ *uniquely into* $\ell_i(X^2) + X \cdot r_i(X^2)$.
   (b) $\mathcal{V} \to \mathcal{P}$*: a random challenge* $\alpha_i \in \mathbb{F}$.
   (c) $\mathcal{P} \to \mathcal{V}$*:* $\mathsf{MT.Commit}(p_{i+1}|_{L_{i+1}})$ *when* $i \neq \sigma$ *or a constant* $f'_{\sigma+1}$ *when* $i = \sigma$*. To compute this,* $\mathcal{P}$ *computes* $p_{i+1}(X) = \ell_i(X) + \alpha_i \cdot r_i(X)$ *from* $f'_i(X)$*, and generates the commitment to* p_{i+1}*.* $\mathcal{P}$ *also computes* $f'_{i+1}(X) = \ell_i(X) + \alpha_i \cdot r_i(X) + \alpha_i^2 \cdot f_{i+1}(X) = p_{i+1}(X) + \alpha_i^2 \cdot f_{i+1}(X)$.
3. $\mathcal{P}$ *and* $\mathcal{V}$ *repeat following procedures* $q = O(\lambda)$ *times:*
   (a) $\mathcal{V} \to \mathcal{P}$*: a random challenge* $\beta \in \mathbb{F}$.
   (b) $\mathcal{P}$ *opens* $\{f_i(\pm\beta^{2^i})\}_{i\in[0,\sigma]}$ *and* $\{p_i(\pm\beta^{2^i})\}_{i\in[\sigma]}$ *via* $\mathsf{MT.Open}$*.* $\mathcal{V}$ *checks them by* $\mathsf{MT.Verify}$*. For each* $i \in [\sigma]$*,* $\mathcal{V}$ *also checks whether the three evaluation pairs* $(\pm\beta^{2^{i-1}}, f'_{i-1}(\pm\beta^{2^{i-1}})), (\alpha_i, p_i(\beta^{2^i}))$ *are on a common line. Note:* $f'_i(\pm\beta^{2^i})$ *can be computed by* $p_i(\pm\beta^{2^i})$ *and* $f_i(\pm\beta^{2^i})$.

A.3 The Rolling Batch FRI

Fast Reed–Solomon interactive oracle proof of proximity (FRI) [7] is a low-degree test for RS codes. Given a multiplicative coset L of size $O(N)$ and a size-N vector $f|_L$, FRI [7] proves that $f|_L$ is δ-close to $\mathrm{RS}[L, N/|L|]$. Here $\delta \in (0, 1)$ is the proximity parameter. FRI has $O(\log N)$ rounds, and in the i-th round, an honest prover sends a Merkle tree commitment to a size-$|L|/2^{i-1}$ codeword, and the verifier sends a random challenge. In the end, the verifier picks q random locations on L and queries at most q entries determined by the picked locations to each committed codeword.

The rolling batch FRI is an FRI variant specified for codewords $f_0|_{L_0}, \ldots, f_\sigma|_{L_\sigma}$ where for $i \in [0, \sigma]$, it holds that $f_i|_{L_i} \in \mathrm{RS}[L_i, \rho]$ and $L_{i+1} = \{x^2 | x \in L_i\}$. We recall the rolling batch FRI in Protocol 3.

B Proof of Theorem 3.1

Proof. We first prove the correctness of Equation (1).

Define $d_v = \log m$, $d_w = \log \ell$ and $d = d_v + d_w$. Define a function $b_k(i) \in \{0, 1\}$ to be the k-th entry of the binary representation of i, where $i \in [N]$. Similarly, define a function

$c_k(a) \in \{0,1\}$ to be the k-th entry of the binary representation of a, where $a \in [m]$. Define a function $d_k(j) \in \{0,1\}$ to be the k-th entry of the binary representation of j, where $j \in [\ell]$.

We next describe entries of $\boldsymbol{v}$ and $\boldsymbol{w}$ explicitly. By definition, we have the a-th entry of $\boldsymbol{v}$ and the j-th entry of $\boldsymbol{w}$ are

$$v_a = \prod_{k=1}^{d_v} x_k^{c_k(a)}, \quad w_j = \prod_{k=1}^{d_w} x_{k+d_v}^{d_k(j)}.$$

For each $j \in [\ell]$, we re-describe the evaluation of $\tilde{f}_j(\boldsymbol{v})$ as

$$\tilde{f}_j(\boldsymbol{v}) = \sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot \prod_{k=1}^{d_v} x_k^{c_k(a)}.$$

By definition, the evaluation of $\tilde{f}(\boldsymbol{x})$ can be represented as

$$y = \tilde{f}(\boldsymbol{x}) = \sum_{i=1}^{N} f_i \cdot \prod_{k=1}^{d} x_k^{b_k(i)}.$$

Representing i as $i = (j-1)\cdot m + a$, we further have

$$y = \sum_{j=1}^{\ell} \sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot \prod_{k=1}^{d} x_k^{b_k(i)}. \tag{3}$$

According to the binary functions, we have when $1 \le k \le d_v$, it holds that $b_k(i) = c_k(a)$. When $d_v + 1 \le k \le d$, it holds that $b_k(i) = d_{k-d_v}(j)$. Hence, we have

$$\prod_{k=1}^{d} x_k^{b_k(i)} = \left(\prod_{k=1}^{d_v} x_k^{c_k(a)}\right) \cdot \left(\prod_{k=1}^{d_w} x_{k+d_v}^{d_k(j)}\right) = v_a \cdot w_j. \tag{4}$$

Combining Equation (3) and Equation (4), we have

$$y = \sum_{j=1}^{\ell} \sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot v_a \cdot w_j. \tag{5}$$

Exchanging the sum sign, Equation (5) is represented as

$$y = \sum_{j=1}^{l} w_j \left(\sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot v_a \right) = \sum_{j=1}^{l} w_j \cdot \tilde{f}_j(\boldsymbol{v}).$$

This exactly proves the validity of Equation (1).

Completeness and Binding. Completeness directly follows from that of PC_ℓ. For the binding property, if there exist $\tilde{f} \neq \tilde{f}'$ such that $\mathsf{VerPoly}(\mathsf{pp}, C, \tilde{f}) = \mathsf{VerPoly}(\mathsf{pp}, C, \tilde{f}') = 1$, then there must exist some $j \in [\ell]$ and $\tilde{f}_j \neq \tilde{f}'_j$, s.t. $\mathsf{VerPoly}(\mathsf{pp}_\ell, C_j, \tilde{f}_j) = \mathsf{VerPoly}(\mathsf{pp}_\ell, C_j, \tilde{f}'_j) = 1$. This means the PCS PC_ℓ is not binding, which is a contradiction. Polynomial binding hence follows.

Knowledge Soundness. Suppose $\mathcal{E}_\ell$ is the extractor of Eval_ℓ and the expected running time is $\mathsf{poly}(m)$. $\mathcal{E}$ runs $\mathcal{E}_\ell$ sequentially and obtains $\tilde{f}_j$ such that $\tilde{f}_j(x_1, \ldots, x_k) = u_j$ for all $j \in [\ell]$. $\mathcal{E}$ then concatenates the coefficients of $\{\tilde{f}_j\}_{j\in[\ell]}$ and outputs the multilinear polynomial $\tilde{f}$ such that $\tilde{\boldsymbol{f}} = (\tilde{\boldsymbol{f}}_1, \ldots, \tilde{\boldsymbol{f}}_\ell)$. Further, the running time is $\ell \cdot \mathsf{poly}(m) = \mathsf{poly}(N)$. Knowledge soundness hence follows.

Complexity. Commitments $C_1, \ldots, C_\ell$, each for a size-$O(m)$ polynomial in $\tilde{f}_1, \ldots, \tilde{f}_\ell$, can be independently generated in parallel. The total commitment complexity is $\ell \times t_{\mathsf{C}}(m)$, or $O(N \log m)$ if $t_{\mathsf{C}}(m) = O(m \log m)$. The opening complexity, verifier complexity, and proof size hold by setting the concrete values of m and ℓ in Definition 3.1. Specifically, if the proof size is $\log^c N$ for a size-N polynomial, the total proof size is $O(\log^c(N/\log^c N) + \log^c N) = O(\log^c N)$. The verifier complexity is similar. As the prover handles size-m polynomials, the size of SRS, if required, is m. □

Protocol 4 ($\mathsf{PIP}_{\mathbf{KZG}}$: sublinear-SRS PCS from pairing).

- $(\mathsf{pp}, \mathsf{srs}) \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$. $\mathsf{pp} = (m, \ell)$ *satisfying* $m\ell = N$, *and* $\mathsf{srs} = \mathsf{Gen}_b(1^\lambda, 2, \log m)$.
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f}, \mathsf{srs})$. *Given* $\tilde{f}$, *the prover* $\mathcal{P}$ *does:*
  1. *Arrange* $\tilde{f}$*'s coefficient vector* $\boldsymbol{f}$ *as an* $m \times \ell$ *matrix* U, *where for every* $j \in [\ell]$, *the* j*-th column is* $U[j] = (f_{(j-1)\cdot m+1}, f_{(j-1)\cdot m+2}, \ldots, f_{j\cdot m})$.
  2. *Treat* $U[j]$ *as the coefficient vector of a size-*m *multilinear polynomial* $\tilde{f}_j$.
  3. *Output* $C \leftarrow (C_1, \ldots, C_\ell)$, *where* $C_j \leftarrow \mathsf{Com}_b(\mathsf{srs}, \tilde{f}_j)$ *for* $j \in [1, \ell]$.
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{srs}, C, x, y; \tilde{f})$. *Given the evaluation point* $\boldsymbol{x}$ *and evaluation* $y = \tilde{f}(\boldsymbol{x})$, $\mathcal{P}$ *and the verifier* $\mathcal{V}$ *do:*
  1. $\mathcal{P} \to \mathcal{V}$: $C = (C_1, \ldots, C_\ell)$.
  2. $\mathcal{V}$: *computes public vectors* $\boldsymbol{v} = \otimes_{k\in[\log m]}(1, x_k)$, $\boldsymbol{w} = \otimes_{k\in[\log \ell]}(1, x_{k+\log m})$, *and* $C' = \prod_{j\in[\ell]} C_j^{w_j}$, *which is an mKZG commitment to* $\tilde{f}' = \sum_{j\in[\ell]} w_j \cdot \tilde{f}_j$ *by the additive homomorphism.*
  3. $\mathcal{P}$ *and* $\mathcal{V}$: *invoke* $\mathsf{Eval}_b(C', \tilde{f}', \mathsf{srs}; y, (x_1, \ldots, x_{\log m}))$ *to prove the validity of the virtual polynomial evaluation* $\tilde{f}'(x_1, \ldots, x_{\log m})$.
  4. $\mathcal{V}$: *accepts iff* $\mathsf{Eval}_b = 1$.

C Construction of $\mathsf{PIP}_{\mathbf{KZG}}$

We construct $\mathsf{PIP}_{\mathsf{KZG}}$ in Protocol 4 by instantiating Protocol 1 with mKZG $(\mathsf{Gen}_b, \mathsf{Com}_b, \mathsf{Eval}_b)$. Let $\tilde{f}$ be a size-$m\ell$ multilinear polynomial partitioned into ℓ sub-polynomials $\tilde{f}^{(1)}, \ldots, \tilde{f}^{(\ell)}$, each of size m. Thanks to mKZG's homomorphism, $\mathcal{P}$ can send only the commitments $\mathsf{Com}(\tilde{f}^{(1)}), \ldots, \mathsf{Com}(\tilde{f}^{(\ell)})$, and avoids sending the sub-polynomial evaluations $\tilde{f}^{(1)}(\boldsymbol{x}), \ldots, \tilde{f}^{(\ell)}(\boldsymbol{x})$. $\mathcal{V}$ constructs a virtual commitment to $\tilde{f}$ by $\prod_{i\in[\ell]} \mathsf{Com}(\tilde{f}^{(i)})^{w_i}$.

We next give a proof for Theorem C.1.

Theorem C.1. *Protocol 4 is an MLPCS secure in the algebraic group model (AGM). The SRS size is $O(m)$, the committing complexity is $O(N)$, the opening complexity is $O(m)\,\mathbb{G}+O(N)\,\mathbb{F}_{\mathsf{rlc}}$, the verifier complexity is $O(\log m)\,P+O(\ell)\,\mathbb{G}$, and the proof size is $O(\ell+\log m)$.*

Proof of Theorem C.1. **Security.** Completeness holds directly by Theorem 3.1. We mainly argue the knowledge soundness by proving an efficient $\mathcal{E}$ such that for any algebraic adversary $\mathcal{A}$ and any choice of $m\ell = \mathsf{poly}(\lambda)$, the probability of $\mathcal{A}$ winning is $\mathsf{negl}(\lambda)$ over the randomness of $\mathcal{A}$, $\mathcal{V}$, and Gen [12]. Here, an algebraic adversary $\mathcal{A}$ refers to the existence of a PPT algorithm that, whenever $\mathcal{A}$ outputs an element $A \in \mathbb{G}$, it also outputs a vector $\boldsymbol{a}$ on field $\mathbb{F}$ such that $A = \langle \boldsymbol{a}, \mathsf{srs}\rangle$. Specifically, the game is as follows [12, 30]:

1. Given $m\ell$ and srs, $\mathcal{A}$ outputs C.
2. $\mathcal{A}$ outputs $\boldsymbol{x}$ and y.
3. $\mathcal{A}$ takes the part of $\mathcal{P}$ in Eval with inputs C, $\boldsymbol{x}$, and y.
4. $\mathcal{E}$, given $\mathcal{A}$'s prior message, outputs $\tilde{f}$ after Step 1 in Eval.
5. $\mathcal{A}$ wins if $\mathcal{V}$ outputs 1 in Eval and $\tilde{f}(\boldsymbol{x}) \neq y$.

An adversary that outputs a commitment C under the AGM is required to additionally output scalar coefficients $a_1, \ldots, a_{n-1}$, which "explain" C as a linear combination of the group elements in the SRS. In Protocol 4, this means that the adversary would output $\tilde{f}'_1, \ldots, \tilde{f}'_\ell$ when it outputs $C_1, \ldots, C_\ell$. Then, the prover and verifier invoke mKZG for a size-m polynomial $\tilde{f}'$. mKZG satisfies evaluation binding [20, 55], *i.e.*, for any point $\boldsymbol{x}$ and corresponding commitment C, no efficient adversary can produce valid proofs that open C to different lists of values at $\boldsymbol{x}$. By the evaluation binding of mKZG to $\tilde{f}' = \sum_{j\in[\ell]} w_j \cdot \tilde{f}'_j$ and its commitment, with a high probability that $\tilde{f}'$ is the correct polynomial with the claimed evaluation y. The knowledge soundness hence follows.

Complexity. The SRS size, committing complexity, and proof size are held directly by Theorem 3.1 and the complexities of the original mKZG. For the opening complexity, the prover first computes the target size-m polynomial $\tilde{f}'$ and then invokes the opening algorithm of mKZG for this polynomial. The first part takes at most $O(N)\,\mathbb{F}_{\mathsf{rlc}}$, and the second part costs $O(m)\,\mathbb{G}$. For the verifier complexity, the verifier first computes the commitment to the target polynomial, which costs $O(\ell)\,\mathbb{G}$. The verifier then invokes the verification algorithm of mKZG for a size-m polynomial, which costs $O(\log m)\,\mathcal{P}$. The complexities hence follow. □

D Security Analysis for zk-PIP$_{\mathbf{FRI}}$

We prove completeness, polynomial binding, knowledge soundness, and zero knowledge for PIP$_{\mathrm{FRI}}$.

Proof. **Completeness** follows from that of the rolling batch FRI, the linearity of RS code, and the equivalence between the multilinear polynomial evaluation and the FRI-like scheme in Step 3, Protocol 2. Specifically, we distill the equivalence between the multilinear polynomial evaluation and the FRI-like scheme from PolyFRIM [60] as Lemma 1 below.

Lemma 1 ([60]). *For any size-$m = 2^{\sigma+1}$ multilinear polynomial $\tilde{f}_0$ and any evaluation point $\boldsymbol{x} = (x_1, \ldots, x_\sigma, x_{\sigma+1})$, following Step 3 of Protocol 2 to fold $\tilde{f}_0$ to obtain the constant $\hat{f}_{\sigma+1}$, it holds that $\hat{f}_{\sigma+1} = \tilde{f}_0(x_1, \ldots, x_\sigma, x_{\sigma+1})$.*

Based on Lemma 1, after setting $x_{\sigma+1} = 0$, we have for any $\boldsymbol{x}$, $\tilde{f}_0(\boldsymbol{x}) = \sum_{j\in[\ell]} w_j \cdot \tilde{f}^{(j)}(\boldsymbol{x}) + \alpha\tilde{s}(\boldsymbol{x})$. The completeness follows since, by the construction of Protocol 1, we have

$$\sum_{j\in[\ell]} w_j \cdot \tilde{f}^{(j)}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{u}\rangle = \tilde{f}(x_1, \ldots, x_\mu) = y.$$

Completeness hence follows.

Polynomial Binding. Given the committed vectors $\hat{f}^{(1)}|_{L_0}$, $\ldots$, $\hat{f}^{(\ell)}|_{L_0}$, the verifier checks their consistency with the Merkle tree commitment. Due to the collision resistance of hash functions underlying Merkle trees, these committed vectors are unique. The verifier then uses efficient RS decoding algorithms to obtain polynomials $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$. Setting the proximity parameter $\delta = (1-\rho)/2$, which is no more than the unique decoding radius of the RS code, these polynomials are unique due to the properties of the RS code. Hence, the target polynomial $\tilde{f}$ computed from these polynomials is also unique. The polynomial binding hence follows.

We prove polynomial binding in the unique decoding radius for simplicity, *i.e.*, the proximity parameter $\delta = (1-\rho)/2$. It can be extended to the list decoding radius using the techniques in DEEP-FRI [9] without much efficiency loss. Intuitively, in the list decoding radius, there could be a list of valid polynomials close enough to the committed codeword. Using the techniques in DEEP-FRI can reduce the list size to one. We kindly refer to [9, 34] for details.

Knowledge Soundness. PIP$_{\mathrm{FRI}}$ is an argument of knowledge in the random oracle model realizing the extractability of Merkle trees, *i.e.*, an extractor can efficiently obtain the committed Merkle leaves [59]. We kindly refer to [59, 60] for the extractability proof. Further, the committed RS codewords can be efficiently decoded by algorithms such as Berlekamp–Welch. After extracting the committed codewords $\hat{f}^{(1)}|_{L_0}, \ldots, \hat{f}^{(\ell)}|_{L_0}$, the extractor outputs ℓ unique polynomials by efficient decoding. It next parses the coefficients of these polynomials, drops the right half of each, and builds the target polynomial $\sum_{j=1}^{\ell} w_j \cdot \tilde{f}^{(j)}$.

Now, we assume a multilinear polynomial $\tilde{f}$ is extracted. We next prove: if $\tilde{f}(\boldsymbol{x}) \neq y$, the verifier accepts with $\mathsf{negl}(\lambda)$.

• Case 1. Some $\hat{f}_i$ for $i \in [0, \mu+1]$ is of size $> 2^{\mu+1-i}$. We argue that the soundness error of this case is $|L_0|/|\mathbb{F}| + ((1+\rho)/2)^{q'} + \mathsf{negl}(\lambda)$. This follows from the soundness of rolling batch FRI according to PolyFRIM. Intuitively, the only difference between rolling batch FRI and our LDT is that in the first round, the input codeword of rolling batch FRI is single,

while the inputs of our LDT are multiple codewords. Still, according to the generalized correlated agreement over span spaces [8]: if $\sum_{i\in[\ell]} \alpha^i \cdot \boldsymbol{u_i}$ is close to a codeword $\boldsymbol{v} \in \mathrm{RS}[\rho, L_0]$ for vectors $\boldsymbol{u_1}, \ldots, \boldsymbol{u_\ell} \in \mathbb{F}^{|L_0|}$ and random challenge α, with a high probability that each vector $\boldsymbol{u_i}$ is close to some codeword belonging to $\mathrm{RS}[\rho, L_0]$. Here, the word "close" means the Hamming distance. Further, the repeated entries between each vector and its corresponding codeword are the same.

• Case 2. The degree of $\{\hat{f}_i\}$ are at most $2^{\mu+1-i}$ for all $i \in [0, \mu+1]$ and the evaluation proofs are all valid. As $\tilde{f}(\boldsymbol{x}) \neq y$, $\tilde{f}(\boldsymbol{x}) + \alpha \cdot \tilde{s}(\boldsymbol{x}, 0) = y + \alpha \cdot y_s$ holds with probability bounded by $1/|\mathbb{F}|$ due to the randomness of α. Otherwise, there must be some i such that $\hat{f}_i(X) \neq \hat{g}_{i-1}(X) + x_i \cdot \hat{h}_{i-1}(X)$, but for query β, the equation holds. According to the Schwartz–Zippel lemma, the soundness error is bounded by $2m/|L_0|$ as the degree of $\hat{f}_i$ is at most $2m$. If picking q independent queries, this error is $(2m/|L_0|)^q$.

Using the union bound argument, the soundness error is $\mathsf{negl}(\lambda) + |L_0|/|\mathbb{F}| + (2^{\mu+1}/|L_0|)^q + ((1+\rho)/2)^{q'}$. By appropriate choices of parameters, this error can be negligible. If the code rate is large, our protocol may lead to a large proof size due to the large q. We can then modify the consistency check in Protocol 2 such that the verifier does not pick β from L_0 but from $\mathbb{F}$. To complete the consistency check, the prover and verifier run a batch FRI-PCS instead to prove the validity of $\{\hat{f}_i(\pm\beta^{2^i})\}_{i\in[0,\sigma]}$. This FRI-PCS can be batched with the rolling batch FRI, introducing a slight additional overhead. The soundness error of this case reduces to $2m/|\mathbb{F}|$.

Like the proof for polynomial binding, we only prove the soundness error in the unique decoding radius. Using the techniques in DEEP-FRI [9], the proof can be extended into the listing decoding radius with a slight additional overhead. We kindly refer to [9, 34, 37] for details.

We next formally prove the soundness of Case 1. Below, we adapt some notations from PolyFRIM. Define functions $\{\mathrm{back}_i\}_{i\in[0,\sigma+1]}$ and sets $\{\mathrm{err}_i\}_{i\in[0,\sigma+1]}$. Define $\mathrm{back}_0(A_0) = A_0$ for $A_0 \subseteq L_0$, and $\mathrm{back}_i(A_i) = \mathrm{back}_{i-1}(A_{i-1})$ for $A_i = \{x^2 | x \in A_{i-1}\}$ and $i \geq 1$. Define $\mathrm{err}_0 = \emptyset$. For $i \in [\sigma+1]$, let the set E be $\{x \in L_{i-1} : \ell_i(x^2) + \alpha r_i(x^2) + \alpha_i^2 \hat{f}_i(x^2) \neq \hat{f}_i'(x^2)\}$, where $\hat{f}_i'(X) = \ell_i(X^2) + X \cdot r_i(X^2)$. Define $\mathrm{err}_i = \mathrm{err}_{i-1} \cup \mathrm{back}_{i-1}(E)$. For $i \in [\sigma+1]$, suppose that $\hat{h}^{(i)}|_{L_i} \in \mathrm{RS}[L_i, \rho]$ and set $A_i = \{x \in L_i | \hat{f}_i'(x) = \hat{h}^{(i)}(x)\}$. Define $\varepsilon_i(\cdot) : L_i \to \{0,1\}$ such that $\varepsilon_i(A_i) = |\mathrm{back}_i(A_i) \cap \mathrm{err}_i|/|L_0| + (1 - |A_i|/|L_i|)$. By definition, $|\mathrm{back}_i(A_i)| = 2^i |A_i|$, $A_i \subseteq L_i$, and $\mathrm{err}_i \subseteq L_i$. Intuitively, $\varepsilon_i(A_i)$ means the probability that the verifier finds inconsistency in the i-th round. Further, $\varepsilon_{\sigma+1}(A_{\sigma+1})$ means the probability that the verifier finds inconsistency and rejects with a single query in Step 5 of Protocol 2. Suppose:

$$\Pr[\varepsilon_{\sigma+1}(A_{\sigma+1}) \geq (1-\rho)/2] \geq 1 - |L_0|/|\mathbb{F}|. \tag{6}$$

That is, with probability at least $1 - |L_0|/|\mathbb{F}|$, the verifier can catch about $\delta = (1-\rho)/2$ inconsistency if it picks only one query. For q' queries, the probability that a malicious prover cheats successfully will be $(1-\delta)^{q'} = ((1+\rho)/2)^{q'}$.

According to PolyFRIM [60, Theorem 3.1], to prove Equation (6), it suffices to prove

$$\Pr[\varepsilon_1(A_1) \geq \delta] \geq 1 - |L_0|/|\mathbb{F}|, \tag{7}$$

with the following cases: (a) For all $i \in [1, \sigma]$, $\Delta(\hat{f}_i'|_{L_i}, \mathrm{RS}[L_i, \rho]) \leq \delta$; (b) For all $i \geq 2$, $\Delta(\hat{f}_i|_{L_i}, \mathrm{RS}[L_i, \rho]) \leq \delta$; (c) At least one of $\Delta(\hat{f}_0'|_{L_0}, \mathrm{RS}[L_0, \rho]) > \delta$ and $\Delta(\hat{f}_1|_{L_1}, \mathrm{RS}[L_1, \rho]) > \delta$ holds.

In the cases above, (a) and (b) are good ones, and we only need to focus on (c). Here, $\hat{f}_0'(X) = \sum_{i=1}^{\ell} \alpha_0^{i-1} \hat{f}^{(i)}(X)$ and $\hat{f}_1'(X) = \hat{f}_0'(X) + \alpha_0^{\ell} \hat{f}_1(X)$. By Lemma 2, for random α_0, with a probability of $1 - |L_0|/|\mathbb{F}|$, it holds that $\Delta(\hat{f}_1'|_{L_1}, \mathrm{RS}[L_1, \rho]) \geq \delta$. The soundness of Case 1 hence follows.

Lemma 2 (The generalized correlated agreement [8])**.** *For $\delta < (1-\rho)/2$, $\{\hat{f}_i\}_{i\in[n-1]}$, multiplicative coset L, and code rate ρ, if*

$$\Pr_{\alpha \leftarrow_\$ \mathbb{F}}[\Delta(\sum_{i=0}^{n-1} \alpha^i \cdot \hat{f}_i|_L, \mathrm{RS}[L, \rho]) \leq \delta] \geq |L|/|\mathbb{F}|,$$

where $\Delta(\boldsymbol{a}, \boldsymbol{b})$ is the relative Hamming distance between $\boldsymbol{a}$ and $\boldsymbol{b}$, there exists $L' \subset L$ and $\hat{p}_i$ with the same degree bounds as $\hat{f}_i$ s.t. $|L'|/|L| \geq 1 - \delta$ and $\hat{f}_i|_{L'} = \hat{p}_i|_{L'}$ for $i \in [0, n-1]$.

Zero Knowledge. Figure 6 gives the simulator. We first note that Protocol 2 runs over univariate $\hat{f}'$ instead of $\tilde{f}'$. To argue zero knowledge, evaluations on $\hat{f}'^{(1)}, \ldots, \hat{f}'^{(\ell)}$ are randomized due to the m random entries to the coefficient of each polynomial. Adding these random entries amounts to adding a size-m random polynomial to $\hat{f}^{(i)}$ with the lower m terms being zero. Hence, evaluations of $\hat{f}'^{(1)}, \ldots, \hat{f}'^{(\ell)}$ on B ($B = O(\lambda) \ll 2^\mu$) are indistinguishable from the real world. After the first round, evaluations on $\hat{f}'^{(i)}$ are masked by $\hat{s}'$; therefore, they are independently and randomly distributed. Note that the simulator need not know α ahead. Hence, even $\boldsymbol{w} = (w_1, \ldots, w_\ell)$ is public, it does not affect zero knowledge. Also, $\mathcal{S}$ runs efficiently as $\tilde{f}_1'$ can be efficiently obtained by solving a linear system with $|B|+1$ constraints and 2^μ variables. □

E Protocol and Proof of Distributed FRI

Protocol 5 presents the formal distributed FRI.

Proof. **Complexity.** We analyze the complexities below.

Prover. Sub-prover $\mathcal{P}_j$ runs FFT to compute $f^{(j)}|_{L_0}$ in $O(m \log m)$ time. In round i, $\mathcal{P}_j$ receives at most $2|L_i|/\ell$ entries to compute $|L_i|/\ell$ entries and to build or open a sub-Merkle tree with $|L_i|/\ell$ leaves, costing $O(\sum_i 2|L_i|/\ell) = O(m/\ell)$. The total sub-prover complexity is $O(m/\ell)$, other

- $C \leftarrow \mathcal{S}_1\left(1^\lambda, \mathsf{pp}\right)$

1. Randomly pick a size-N multilinear polynomial $\tilde{f}'$, split it as ℓ size-m polynomials, and add m random entries to the right end of each polynomial's coefficient. Denote these size-$2m$ univariate polynomials as $\hat{f}'^{(1)}, \ldots, \hat{f}'^{(\ell)}$.
2. Randomly select a size-$2m$ polynomial $\tilde{s}'$.
3. Run $C \leftarrow \mathsf{MT.Com}([\hat{f}'^{(1)}|_{L_0}, \ldots, \hat{f}'^{(\ell)}|_{L_0}, \hat{s}'|_{L_0}])$.

- $\langle \mathcal{S}_2, \mathcal{A} \rangle(\mathsf{pp}, C, \boldsymbol{x})$

1. Send $(y, \tilde{s}'(\boldsymbol{x}, 0))$ to $\mathcal{A}$ given oracle access to $y = \tilde{f}(\boldsymbol{x})$.
2. Receive α from $\mathcal{A}$.
3. Given the query set of both consistency and validity checks $B = \{\beta_j\}_{j \in [Q]}$, $\mathcal{S}_2$ generates a size-m polynomial $\tilde{f}'_1$ s.t. Let $\hat{f}'_1$ be the corresponding univariate polynomial of $\tilde{f}'_1$. Let $\hat{f}'_0 \leftarrow \sum_{i=1}^{\ell} w_i \hat{f}'^{(i)}$ for $\boldsymbol{w} = \otimes_{\ell \in [\log \ell]}(1, x_{\ell + \log m})$.
   - For $\beta_j \in B$, set

$$\hat{f}'_1(\beta_j^2) = \frac{\hat{f}'_0(\beta_j) + \alpha \tilde{s}'(\beta_j) + \hat{f}'_0(-\beta_j) + \alpha \tilde{s}'(-\beta_j)}{2} + x_1 \cdot \frac{\hat{f}'_0(\beta_j) + \alpha \tilde{s}'(\beta_j) - \hat{f}'_0(-\beta_j) - \alpha \tilde{s}'(-\beta_j)}{2\beta_j}.$$

   - $\tilde{f}'_1(x_2, x_3, \ldots, x_{\log m}, 0) = y$.

   $\mathcal{S}_2$ opens $\hat{f}'^{(1)}, \ldots, \hat{f}'^{(\ell)}$ to open $\hat{f}'_0$ honestly, sets $\hat{f}'_1$ as the second-round polynomial, and acts honestly thereafter.

Figure 6: Simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ of zk-PIP_{FRI}

than the $O(m \log m)$ in the distributed commitment. For the master prover $\mathcal{P}_0$, receiving ℓ roots to build a Merkle tree in each of σ rounds costs $O(\ell)$ time. In the last $\log \ell$ rounds, $\mathcal{P}_0$ acts as a non-distributed prover, the time cost is $O(\ell)$. Overall, the prover complexity of $\mathcal{P}_0$ is $O(\ell\sigma) = O(\ell \log m)$.

Proof size and verifier complexity. From the verifier's view, the distributed FRI is the same as a non-distributed batch FRI with ℓ size-$O(m)$ codewords $f^{(1)}|_{L_0}, \ldots, f^{(\ell)}|_{L_0}$ as inputs. In the first round, the verifier receives $\ell \cdot \lambda$ entries and checks their verification path of a Merkle tree with $O(m)$ leaves committed via our optimized Merkle tree method, which costs $O(\lambda\ell)\mathbb{F} + O(\lambda)\mathsf{H}_\ell$ for proof size and verifier complexity. In each of the following rounds, the verifier receives λ entries and checks their verification path, which costs $O(\lambda)\mathbb{F} + O(\log m)\mathsf{H}$, and $O(\lambda \log m)\mathbb{F} + O(\log^2 m)\mathsf{H}$ for $\log m$ rounds. Hence, the total proof size and verifier complexity follow.

Communication complexity. In round 1, sub-prover $\mathcal{P}_j$ sends ℓ size-$O(m/\ell)$ codewords to $\mathcal{P}_k$, so the total communication is $O(m\ell)$. In round i, $\mathcal{P}_j$ receives at most $2|L_i|/\ell$ entries, costing an amortized $O(\sum_i |L_i|/\ell) = O(m/\ell)$ communication for all rounds, and $O(m)$ for all provers. $\mathcal{P}_0$ receives ℓ roots

Protocol 5 (Distributed FRI). *Suppose master prover $\mathcal{P}_0$ and ℓ sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$. Secret inputs are ℓ size-m polynomials $f^{(1)}, \ldots, f^{(\ell)}$. Public inputs include a size-$O(m)$ multiplicative coset L_0. For $i \in [0, \log m]$, define $L_i = \{x^2 | x \in L_{i-1}\}$ and $d_i = |L_i|/\ell$. To prove $f^{(j)}|_{L_0} \in \text{RS}[L_0, \rho]$ for all $j \in [\ell]$, $\mathcal{P}$ and $\mathcal{V}$ run:*

1. ***(Commit)*** *$\mathcal{P}_0 \to \mathcal{V}$:* $\mathsf{MT.Commit}([f^{(1)}|_{L_0}, \ldots, f^{(\ell)}|_{L_0}])$.
   (a) *$\mathcal{P}_j$: computes $f^{(j)}|_{L_0}$ and sends the blocks $f^{(j)}|_{L_0}[(k-1)d_0 + 1], \ldots, f^{(j)}|_{L_0}[kd_0]$ to $\mathcal{P}_k$, $\forall k \in [\ell] \setminus \{j\}$.*
   (b) *$\mathcal{P}_j \to \mathcal{P}_0$: a Merkle sub-tree root $\mathsf{H}_0^{(j)}$. The tree has d_0 leaves, and the n-th leaf is $f^{(1)}|_{L_0}[(j-1)d_0 + n] || f^{(2)}|_{L_0}[(j-1)d_0 + n] || \cdots || f^{(\ell)}|_{L_0}[(j-1)d_0 + n]$.*
   (c) *$\mathcal{P}_j$: computes $f|_{L_0}[h] = \sum_{j \in [\ell]} \alpha^j \cdot f^{(j)}|_{L_0}[h]$ for all $h \in [(j-1)d_0 + 1, jd_0]$. This is used in the opening phase.*
   (d) *$\mathcal{P}_0$: forms the final commitment C from $\mathsf{H}_0^{(1)}, \ldots, \mathsf{H}_0^{(\ell)}$ by treating each $\mathsf{H}_0^{(j)}$ as a sub-root. $\mathcal{P}_0$ sends C to $\mathcal{V}$.*
2. ***(Challenge Assignment)*** *$\mathcal{V} \to \mathcal{P}_0$: random $\alpha \in \mathbb{F}$ for batch FRI. $\mathcal{P}_0$ assigns α to each $\mathcal{P}_j$.*
3. ***(Open)*** *For $i \in [\sigma]$, $\sigma = \log(m/\ell)$:*
   (a) *$\mathcal{V} \to \mathcal{P}_0$: random $\alpha_i \in \mathbb{F}$. $\mathcal{P}_0$ assigns α_i to each $\mathcal{P}_j$.*
   (b) *$\mathcal{P}_0 \to \mathcal{V}$: a Merkle tree commitment C_i to $f_i|_{L_i}$.*
      i. *$\mathcal{P}_j$: computes the j-th part of $f_i|_{L_i}$, i.e., $\{f_i|_{L_i}[h]\}$ for $h \in [(j-1)d_i + 1, jd_i]$. To compute this, $\mathcal{P}_j$ needs two entries on $f_{i-1}|_{L_{i-1}}$ with fixed locations. As all provers hold $f_{i-1}|_{L_{i-1}}$, $\mathcal{P}_j$ can always obtain these entries. $\mathcal{P}_j$ then computes $f_i|_{L_i}[h]$ by $F(f_{i-1}|_{L_{i-1}}[h], f_{i-1}|_{L_{i-1}}[h + |L_{i-1}|/2], \alpha_i)$.*
      ii. *$\mathcal{P}_j \to \mathcal{P}_0$: a Merkle tree root $\mathsf{H}_i^{(j)}$ to $\{f_i|_{L_i}[h]\}$ for $h \in [(j-1)d_i + 1, jd_i]$. When $i = \log(m/\ell)$, $\mathcal{P}_j$ sends $f_i|_{L_i}[jd_i]$ to $\mathcal{P}_0$.*
      iii. *$\mathcal{P}_0$: computes C_i from $\mathsf{H}_i^{(1)}, \ldots, \mathsf{H}_i^{(\ell)}$.*
4. ***(Local Open)*** *For $i \in [\sigma + 1, \log m + 1]$, $\mathcal{P}_0$ and $\mathcal{V}$ run the following procedures non-distributedly:*
   (a) *$\mathcal{V} \to \mathcal{P}_0$: random challenge $\alpha_i \in \mathbb{F}$.*
   (b) *$\mathcal{P}_0 \to \mathcal{V}$: $\mathcal{P}_0$ locally folds $f_{i-1}|_{L_{i-1}}$ to compute $f_i|_{L_i}$ and commits via a Merkle tree.*
5. ***(Verification)*** *Repeat the following for $q = O(\lambda)$ times:*
   (a) *$\mathcal{V} \to \mathcal{P}_0$: random query $\beta \in L_0$. This means that $\mathcal{V}$ needs to query $f_i(\pm\beta^{2^i})$ for all $i \in [0, \log m]$.*
   (b) *$\mathcal{P}_j \to \mathcal{P}_0$: For $i \in [\log(m/\ell)]$, if $f_i(\pm\beta^{2^i})$ corresponds to root $\mathsf{H}_i^{(j)}$, $\mathcal{P}_j$ sends values and tree path to $\mathcal{P}_0$.*
   (c) *$\mathcal{P}_0 \to \mathcal{V}$: For last $\log \ell$ rounds, $\mathcal{P}_0$ opens C_i herself.*
   (d) *$\mathcal{V}$: verifies Merkle trees by* $\mathsf{MT.Verify}$. *Check if $2f_i(\beta^{2^i}) = f_{i-1}(\beta^{2^{i-1}}) + f_{i-1}(-\beta^{2^{i-1}}) + \alpha_i/\beta^{2^{i-1}} \cdot (f_{i-1}(\beta^{2^{i-1}}) - f_{i-1}(-\beta^{2^{i-1}}))$. Accept iff all pass.*

Protocol 6 (DEPIP$_{\text{FRI}}$). *Suppose master prover $\mathcal{P}_0$ and sub-provers $\mathcal{P}_1,\ldots,\mathcal{P}_\ell$.* DEPIP$_{\text{FRI}}$ *has the following algorithms:*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$: pp *includes* $\mathbb{F}$, *power-of-two integers* $\ell = O(\log N)$, $m = N/\ell$, *and a multiplicative coset* L_0. *Let* $\sigma = \log m$. *For* $i \in [\sigma]$, *define* $L_i = \{x^2 | x \in L_{i-1}\}$.
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f}, \tilde{f}^{(1)}, \ldots, \tilde{f}^{(\ell)})$: *Given the multilinear polynomial* $\tilde{f}$, *split it into* ℓ *size-m multilinear polynomials* $\tilde{f}^{(1)},\ldots,\tilde{f}^{(\ell)}$ *by Protocol 1. Assume* $\mathcal{P}_j$ *is assigned with* $\tilde{f}^{(j)}$, $\mathcal{P}_1,\ldots,\mathcal{P}_\ell$ *invoke Step 1 of the distributed FRI to generate the commitment* C *to* $\tilde{f}^{(1)},\ldots,\tilde{f}^{(\ell)}$.
- $b \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, [\hat{f}^{(1)}|_{L_0}, \ldots \hat{f}^{(\ell)}|_{L_0}, \tilde{f})$. *This algorithm is the same as that of* PIP$_{\text{FRI}}$, *and we omit it here.*
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; (\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}))$.

  1. $\mathcal{P}_0$ *and sub-provers invoke Steps 3 and 4 of the distributed FRI to generate* $\hat{f}_1,\ldots,\hat{f}_\sigma$ *and their commitments, as well as the constant polynomial* $\hat{f}_{\sigma+1}$. *The* $\hat{f}_1,\ldots,\hat{f}_\sigma$ *are generated in the same way as Step 3,* Eval *in* PIP$_{\text{FRI}}$, *except with the following changes: (1) Modify* α^j *to* w_j *for* $j \in [\ell]$, *where* $\boldsymbol{w} = \otimes_{k\in[\log \ell]}(1, x_{k+\log m})$; *(2) Modify* α_i *to* x_i *for* $i \in [\sigma]$.
     At the end, $\mathcal{P}_j$ *has the* j*-th part of* $\hat{f}_i|_{L_i}$.
  2. $\mathcal{P}_0$ *and sub-provers prove the low-degree properties of* $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ *as well as* $\hat{f}_1,\ldots,\hat{f}_\sigma$. *To prove the former, the provers invoke Steps 3 and 4 of the distributed FRI. To prove the latter, the provers modify the above steps to run a distributed rolling batch FRI for additionally proving the low-degree properties of* $\hat{f}_1,\ldots,\hat{f}_\sigma$. *The modifications are as follows.*
     In the i*-th round,* $\mathcal{P}_j$ *holds the* j*-th part of* $\hat{f}_{i-1}|_{L_{i-1}}$ *and* $f'_{i-1}|_{L_{i-1}}$. *The latter is from the* $i-1$*-th folding of rolling batch FRI. When* $i = 1$, $f'_0|_{L_0} = \sum_{j\in[\ell]} \alpha^j \cdot \hat{f}^{(j)}|_{L_0}$.
     $\mathcal{P}_j$ *follows the distributed FRI to compute the* j*-th part of* $f_i|_{L_i}$ *by folding* $f'_{i-1}|_{L_{i-1}}$ *locally.* $\mathcal{P}_j$ *then computes the* j*-th part of* $f'_i|_{L_i}$ *according to Equation (2).*
  3. $\mathcal{V}$ *queries* $q = O(\lambda)$ *entries to check the consistency among* $\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_\sigma$, *where* $\hat{f}_0(X) = \sum_{j\in[\ell]} w_j \cdot \hat{f}^{(j)}(X)$. $\mathcal{V}$ *also queries* $q' = O(\lambda)$ *entries to check the low-degree properties of* $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ *and* $\hat{f}_1,\ldots,\hat{f}_\sigma$. *From the verifier's view, these queries and the verification algorithms are the same as* PIP$_{\text{FRI}}$.
     To open these queries, the Merkle tree opening algorithms follow from Step 5 of the distributed FRI.

in each round, amounting to $O(\ell \log m)$. Hence, the total communication complexity is $O(m\ell)$.

Security. As with several other distributed PCSs [40, 41, 54], we assume each sub-prover is honest here. We discuss the accountability of DEPIP$_{\text{FRI}}$ in Section 5.3.

As stated in the complexity analysis of proof size, from the verifier's view, the transcripts he receives are the same as those of a batch FRI [8]. Hence, the security properties directly follow from the batch FRI. □

F Construction of DEPIP$_{\text{FRI}}$

We present the formal algorithms of DEPIP$_{\text{FRI}}$ in Protocol 6.

In the Com algorithm, each prover initially possesses a sub-polynomial derived from the target polynomial $\tilde{f}$. The provers then invoke the distributed commitment of distributed FRI with the twisted univariate polynomial representations of their respective sub-polynomials as inputs.

In Step 1 of the Eval protocol, the provers invoke a variant of the distributed evaluation of distributed FRI to generate folded polynomials $\hat{f}_1,\ldots,\hat{f}_\sigma,\hat{f}_{\sigma+1}$ and their commitments for the first σ polynomials. The folding parameters of these folded polynomials are determined by the evaluation point as established in standard PCSs like PIP$_{\text{FRI}}$ and PolyFRIM [60].

Subsequently, in Step 2, the provers invoke a generalized distributed rolling batch FRI protocol to prove the low-degree properties of the input polynomials and folded polynomials.

Finally, in Step 3 of the Eval protocol, the verifier checks the validity of the folded polynomials via a variant of distributed FRI. The verifier also checks the low-degree properties of the input polynomials and folded polynomials.

G Supplementary Experiments

G.1 Detailed Performance Metrics of PIP$_{\text{FRI}}$

Table 11 presents the detailed figures of the eight MLPCSs at polynomial sizes of 2^{22} and 2^{27} in Figure 2. When the polynomial size is 2^{22}, all MLPCSs work well. For the concrete performance, PIP$_{\text{FRI-1/8}}$ has 50× faster committing time, 30× faster opening time, and 10× smaller memory costs than DeepFold. Its verifier time and proof size remain competitive. Also, PIP$_{\text{FRI-1/8}}$ has nearly the same prover time with Orion, but with 30× better verifier time and proof size. Lastly, PIP$_{\text{FRI-1/2}}$ outperforms all listed MLPCSs in committing time, opening time, and memory costs.

When the polynomial size is 2^{27}, only Hyrax, PIP$_{\text{FRI-1/8}}$, and PIP$_{\text{FRI-1/2}}$ can work, and "–" means missing of data. For concrete performance, PIP$_{\text{FRI-1/8}}$ has the smallest proof size and the fastest verifier time, with the latter 25× faster than Hyrax. PIP$_{\text{FRI-1/2}}$ has the fastest committing or opening time and the lowest memory costs. Its prover (committing + opening) time is 80× faster than Hyrax.

G.2 Evaluation of PIP$_{\text{KZG}}$ and SNARKs

Methodology. We adapt the open-source implementations of mKZG from HyperPlonk [19]. Based on it, we implement PIP$_{\text{KZG}}$. We set ℓ to be 16 and $m = N/\ell$ for size-N polynomials. We also compare the open-source implementations

Table 11: Detailed performance metrics at polynomial sizes 2^{22} or 2^{27} for eight MLPCS schemes in Figure 2

Scheme	Size	Commit	Open	Verify	Proof size	Memory	Size	Commit	Open	Verify	Proof size	Memory
mKZG [45]	2^{22}	15.5 s	16.6 s	12 ms	1.4 KB	2.9 GB	2^{27}	–	–	–	–	–
Hyrax [51]		46.5 s	0.3 s	119.8 ms	64 KB	0.3 GB		1626.2 s	8.3 s	756 ms	384 KB	8.2 GB
Virgo [59]		124.4 s	123.3 s	2.2 ms	274 KB	31.1 GB		–	–	–	–	–
Orion [56]		1.9 s	1.1 s	60.1 ms	7214 KB	0.8 GB		–	–	–	–	–
PolyFRIM [60]		113.8 s	64.3 s	3.1 ms	359 KB	13.2 GB		–	–	–	–	–
DeepFold [34]		115.7 s	33.8 s	1.8 ms	194 KB	9.9 GB		–	–	–	–	–
PIP$_{\text{FRI-1/8}}$		2.3 s	0.9 s	**1.8** ms	**196** KB	0.8 GB		99.9 s	15.4 s	**2.9** ms	**358** KB	20.5 GB
PIP$_{\text{FRI-1/2}}$		**0.6** s	**0.2** s	3.7 ms	371 KB	**0.2** GB		**23.4** s	**3.7** s	**6.2** ms	747 KB	**6.7** GB

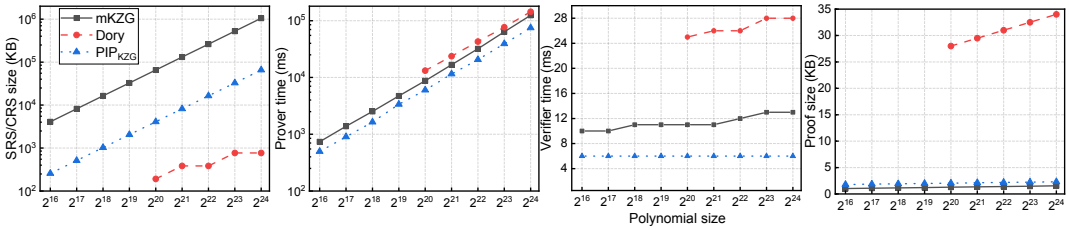

Figure 7: Performance of mKZG, Dory, and PIP$_{\text{KZG}}$

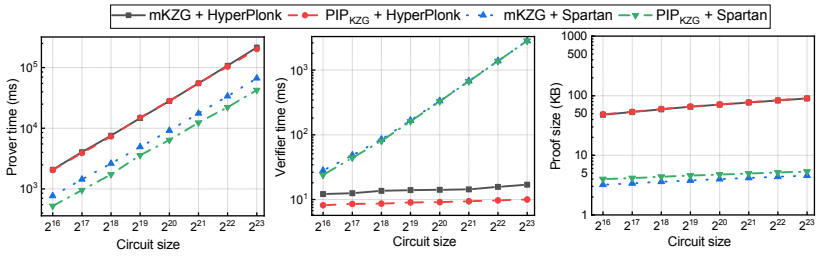

Figure 8: Performance of SNARKs constructed by mKZG or PIP$_{\text{KZG}}$ and PIOPs in Spartan or HyperPlonk

of Dory from HyperPianist [22]. All PCSs use the BN254 elliptic curve and the arkworks ecosystem in Rust.

Performance. Figure 7 shows the performance of mKZG, Dory, and PIP$_{\text{KZG}}$ for multilinear polynomials of sizes from 2^{16} to 2^{24}. We only succeeded in running Dory then the polynomial size exceeds 2^{20}. PIP$_{\text{KZG}}$ shows a tradeoff compared with mKZG. The SRS size of PIP$_{\text{KZG}}$ is 10× shorter, the prover time is 1.7× faster, and the verifier time can be 2× faster. Specifically, committing times are nearly the same, while the opening time of PIP$_{\text{KZG}}$ is 10× faster. The proof size of PIP$_{\text{KZG}}$ is 1.5-1.8× larger than mKZG. Despite this, for size-2^{24} polynomial, the concrete proof size is only 2.2 KB. Besides, in SNARKs such as Spartan and HyperPlonk from multilinear PIOPs, this proof size gap would be reduced after counting the PIOP proof size, as shown below in Figure 8. Compared with the transparent Dory, PIP$_{\text{KZG}}$ requires trusted setups and also has a larger SRS. However, for efficiency, PIP$_{\text{KZG}}$ has a 2× faster prover, 4× faster verifier, and a 10× smaller proof size. This is because the prover and verifier of PIP$_{\text{KZG}}$ runs over the more efficient type-one group instead of the target group as in Dory.

Figure 8 shows the performance of SNARKs using mKZG or PIP$_{\text{KZG}}$ and PIOPs in Spartan [47] or HyperPlonk [19]. The SNARK from "PIP$_{\text{KZG}}$ + HyperPlonk" holds a competitive prover time, verifier time, and proof size compared with "mKZG + HyperPlonk". This is because the HyperPlonk PIOP requires 3 PCS committing algorithms but only 1 opening algorithm, and the PCS committing times of mKZG and PIP$_{\text{KZG}}$ are nearly the same. However, as the SRS is only determined by the underlying PCS, the SRS size of "PIP$_{\text{KZG}}$ + HyperPlonk" is 10× shorter than "mKZG + HyperPlonk". When combined with the Spartan PIOP, "PIP$_{\text{KZG}}$ + Spartan" features a 1.5× faster prover time than "mKZG + Spartan", with a 1.1× larger proof size and a competitive verifier time due to the effect of linear-verifier PIOP.

Table 12: Prover times of Kaizen by Virgo or PIP$_{\text{FRI}}$

Batch Size	$N = 2$		$N = 4$	
Kaizen [2] for LeNet	Virgo	PIP$_{\text{FRI}}$	Virgo	PIP$_{\text{FRI}}$
PCS (s)	15.6	3.4	27.8	5.7
Others (s)	17.8		23.5	
Total (s)	33.4	21.2	51.3	29.2

G.3 Overcoming Prover Time Bottlenecks in Machine Learning with PIP$_{\text{FRI}}$

Table 12 presents the estimated prover times for PCSs of Kaizen and other components from its open-source library [24]. We do not compare with Kaizen using Orion, as experiments show Orion is worse than using Virgo.