DeepFold: Efficient Multilinear Polynomial Commitment from Reed-Solomon Code and Its Application to Zero-knowledge Proofs

Yanpei Guo *
guo.yanpei@u.nus.edu

Xuanming Liu *
hinsliu@zju.edu.cn

Kexi Huang *
Cauchy_0326xz@outlook.com

Wenjie Qu *
wenjiequ@u.nus.edu

Tianyang Tao *
tianyangtao@u.nus.edu

Jiaheng Zhang *
jhzhang@nus.edu.sg

Abstract

This work presents DeepFold, a novel multilinear polynomial commitment scheme (PCS) based on Reed-Solomon code that offers optimal prover time and a more concise proof size. For the first time, DeepFold adapts the FRI-based multilinear PCS to the list decoding radius setting, requiring significantly fewer query repetitions and thereby achieving a 3× reduction in proof size compared to BaseFold (Crypto'24), while preserving its advantages in prover time. Compared with PolyFRIM (USENIX Security'24), DeepFold achieves a 2× improvement in prover time, verifier time, and proof size. Another contribution of this work is a batch evaluation scheme, which enables the FRI-based multilinear PCS to handle polynomials whose size is not a power of two more efficiently.

Our scheme has broad applications in zk-SNARKs, since PCS is a key component in modern zk-SNARK constructions. For example, when replacing the PCS component of Virgo (S&P'20) with DeepFold, our scheme achieves a 2.5× faster prover time when proving the knowledge of a Merkle tree with 256 leaves, while maintaining the similar proof size. When replacing the PCS component of HyperPlonk (Eurocrypt'23) with DeepFold, our scheme has about 3.6× faster prover time. Additionally, when applying our arbitrary length input commitment to verifiable matrix multiplications for matrices of size 1200×768 and 768×2304, which are actual use cases in GPT-2 model, the performance showcases a 2.4× reduction in prover time compared to previous approaches.

1 Introduction

Polynomial Commitment Scheme (PCS) [5, 26] is a powerful cryptographic primitive that allows a prover to commit to a μ-variate polynomial f defined over a field $\mathbb{F}$ with a degree bound of d using a short commitment. Given $\vec{x} \in \mathbb{F}^{\mu}$, the prover can create a proof to convince the verifier that $f(\vec{x}) = y$ for some $y \in \mathbb{F}$. PCS has broad applications like verifiable secret sharing [39, 50], proofs of retrievability [25], and data availability sampling [24, 30]. A PCS is *succinct* if both the proof size and the verifier time are sublinear in the polynomial's size $n = (d+1)^{\mu}$. Succinct PCS is an integral building block of *zero-knowledge succinct non-interactive argument of knowledge* (zk-SNARK) [18, 20, 21, 40], which are particularly useful in various domains, including but not limited to cryptocurrencies [15, 38], blockchain rollup [32, 46], AI regulations [33, 49], and anonymous credentials [37]. A succinct PCS can be integrated with a *Polynomial Interactive Oracle Proof* (PIOP) [9] to construct a zk-SNARK.

The concept and initial construction of univariate PCS (namely, $\mu = 1$) originated from the pioneering work by Kate, Zaverucha, and Goldberg [26] (hereafter, KZG). With the use of pairing groups, one major merit of KZG is its constant verifier time and proof size in the evaluation of a univariate polynomial. However, a long-standing issue that has been widely criticized with KZG is its reliance on a *trusted setup* to generate certain parameters, which often raises security concerns. Another disadvantage of KZG is the high computational overhead of the pairing group operations for the prover. Therefore, there have been numerous studies [5, 22, 47] focusing on creating PCS based on *error-correcting codes* (ECC), which is *transparent* and only relies on lightweight hash functions. Among these schemes, the *Fast Reed-Solomon Interactive Oracle Proof of Proximity* (FRI) [5] is a compelling construction for *univariate* PCS based on Reed-Solomon (RS) codes. FRI is praised for its efficient prover without relying on pairing group operations, as well as its transparent setup. The verifier time and the proof size are poly-logarithmic with respect to the polynomial size. Consequently, it has become an appealing univariate PCS choice for practical applications [6, 8, 23].

FRI-based Multilinear PCS. In recent years, multivariate polynomials ($\mu > 1$), especially multilinear polynomials where the degree in each variable is at most one, have been widely applied in constructing efficient SNARKs [18, 21, 40]. Given its transparency and efficiency, an interesting topic is to use the technique of FRI to construct a PCS for multivariate polynomials, especially multilinear polynomials. How-

[*] National University of Singapore

ever, adapting the (univariate) FRI for multivariate polynomials is non-trivial, because RS codes are originally designed for the univariate one. Several works, such as Virgo [51], PolyFRIM [52], and BaseFold [48], have aimed to achieve this transition. Nonetheless, these schemes always result in a significant increase in prover time or proof size when compared to the univariate FRI protocol, which is not optimal. Therefore, it remains an open question to achieve a multilinear PCS based on FRI with optimal prover time and compact proof size.

FRI **query number.** When constructing PCS based on FRI techniques, an important metric is the *query number*, which determines the final proof size. FRI considers the setting where a verifier is given only oracle access to a vector $\vec{v}$. With the help of an untrusted prover, the verifier needs to distinguish, by querying $\vec{v}$ at a few locations *repeatedly*, whether $\vec{v}$ is a valid RS codeword or whether $\vec{v}$ is far from all codewords in relative Hamming distance. This is called a RS code *interactive oracle proof of proximity* (IOPP) [9]. In the original univariate FRI, the query number is restricted by a limitation known as the *unique decoding radius*. Several works [10, 12, 27] have managed to overcome this limitation and reduce the query number. For example, for an RS code with a code rate of $\frac{1}{8}$, under the unique decoding radius setting, to achieve 100-bit security, the query number exceeds 120, whereas in DEEP-FRI [10], the number is reduced to only 34, resulting in approximately 3× smaller concrete proof size for FRI-based univariate PCS.

Recently, BaseFold [48] discovered a special property of FRI and invoked FRI for a purpose other than RS code IOPP, achieving multilinear PCS with competitive prover time. Its remarkable prover efficiency has made it appealing to numerous industrial projects [19, 41]. However, this unconventional usage of FRI renders previous optimizations for lowering the query number of RS IOPP invalid. Consequently, the query number in BaseFold has reverted to the same level as in the original univariate FRI, resulting in a significant increase in proof size. Concretely, when instantiating BaseFold for a multilinear polynomial with $\mu = 22$ variables, the proof size is 619 KB, whereas the proof size for DEEP-FRI is only about 200 KB for a univariate polynomial degree with 2^{22}. Therefore, it is of significant importance to reduce the proof size of BaseFold while maintaining its prover efficiency.

Arbitrary length input. Another significant drawback of FRI-based PCS is the overhead when encoding inputs of arbitrary length. For instance, when applying a PCS to encode a vector with $2^k + 2^{k/2}$ inputs for some $k \geq 1$, we need to pad the vector to size 2^{k+1} to fit a $(k+1)$-variate multilinear polynomial. If one chooses to commit to the polynomial using a multilinear variant of KZG (mKZG), the overhead from padding is negligible, as it can skip those zero terms directly. However, when using FRI-based PCS, the RS encoding causes the entire 2^{k+1}-sized vector to become non-zero, thereby resulting in nearly double the prover time. This situation is less studied in the literature, but we find it common in real-world scenarios, which often involve inputs of random sizes.

The above issue becomes even more significant for verifiable matrix multiplications [43], because committing to a matrix requires padding in two dimensions, potentially leading to up to 4× overhead. We find this problem very common in *zero-knowledge machine learning* (zkML) [42], which aims to make ML inference verifiable. For example, in GPT-2 [36], the attention matrix is of size 768×2304. When handling verifiable matrix multiplication with such matrices, there is always a need for an efficient PCS to commit to and evaluate a multilinear polynomial encoded from these "arbitrary" matrices. Therefore, these challenges motivate us to develop a new technique to efficiently handle inputs of arbitrary length.

In summary, we propose two open research questions for improving FRI-based multilinear polynomial commitments:

1. Can we build a FRI-based multilinear PCS with optimal efficiency, and push the proof size boundary even lower compared to the state-of-the-art [48]?

2. Can we develop an efficient technique to commit arbitrary-size inputs using FRI without any overhead?

1.1 Contributions

We give affirmative answers to the above questions. Our contributions can be summarized as follows:

- DeepFold**: A multilinear PCS with optimal prover time and smaller proof size.** Our main contribution is DeepFold, an innovative multilinear PCS with $O(n \log n)$ commit time and $O(n)$ evaluating time, where n is the size of the polynomial. The verifier time and the proof size are both $O(s_L \cdot \log^2 n)$, where s_L is the query number under the *list decoding* setting, overcoming *unique decoding* limitation. Concretely, when the code rate is set to $\frac{1}{8}$, the query number s_L is about 34, while the query number in prior work BaseFold [48], s_U, exceeds 100 due to the unique decoding limitation. This translates to more than 3× optimization in proof size. Using FS transformation, DeepFold can be made non-interactive [13].

- **Efficient evaluation for PCS encoding arbitrary length inputs from batching** FRI**.** For the first time in the literature, we study the problem of committing to multilinear polynomials encoding inputs of arbitrary length based on the FRI technique. Instead of padding and committing the inputs as a whole vector, we divide the inputs into several parts and commit them using multiple multilinear polynomials with different numbers of variables. To ensure that this transformation does not significantly increase the overhead of the evaluation, a novel batching evaluation technique is developed, enabling evaluations for different polynomials with varying sizes at one time. This batching technique is also leveraged to non-trivially make DeepFold zero-knowledge.

Table 1: Complexities comparison of μ-variate multilinear polynomial commitments to achieve λ-bit security.

Scheme	Transp.	Assumption	Commit	Evaluate	Verify	Proof Size
mKZG [35]	no	q-SDH	$O(n)\mathbb{G}_B$	$O(n)\mathbb{G}_B$	$O(\log n)\mathbb{G}_B$	$O(\log n)$
DARK [17]	yes/no	Strong RSA	$O(n)\mathbb{G}_U$	$O(n\log n)\mathbb{G}_U$	$O(\log n)\mathbb{G}_U$	$O(\log n)$
Bulletproofs [16]	yes	DLog	$O(n)\mathbb{G}_P$	$O(n)\mathbb{G}_P$	$O(n)\mathbb{G}_P$	$O(\log n)$
Hyrax [44]	yes	DLog	$O(n)\mathbb{G}_P$	$O(n)\mathbb{F}+O(\sqrt{n})\mathbb{G}_P$	$O(\sqrt{n})\mathbb{G}_P$	$O(\sqrt{n})$
Brakedown [22]	yes	$O(n)\mathbb{H}$		$O(n)\mathbb{F}$	$O(\sqrt{n})\mathbb{H}$	$O(\lambda\sqrt{n})$
Orion [47]	yes	$O(n)\mathbb{H}$		$O(n)\mathbb{F}$	$O(s_O\log^2 n)\mathbb{F}$	$O(s_O\log^2 n)$, 3789KB
Virgo [51]	yes	$O(n\log n)\mathbb{F}+O(n)\mathbb{H}$		$O(n\log n)\mathbb{F}+O(n)\mathbb{H}$	$O(s_L\log^2 n)\mathbb{H}$	$O(s_L\log^2 n)$, 320KB
PolyFRIM [52]	yes	$O(n\log n)\mathbb{F}+O(n)\mathbb{H}$		$O(n)\mathbb{H}$	$O(s_L\log^2 n)\mathbb{H}$	$O(s_L\log^2 n)$, 570KB
Basefold [48]	yes	$O(n\log n)\mathbb{F}+O(n)\mathbb{H}$		$O(n)\mathbb{H}$	$O(s_U\log^2 n)\mathbb{H}$	$O(s_U\log^2 n)$, 929KB
DeepFold	yes	$O(n\log n)\mathbb{F}+O(n)\mathbb{H}$		$O(n)\mathbb{H}$	$O(s_L\log^2 n)\mathbb{H}$	$O(s_L\log^2 n)$, 304KB

For $\mathbb{G}_i$, where $i \in \{B,U,P\}$, it denotes a group with a *B*ilinear map, of *U*nknown order, or of known *P*rime order;
For s_i, where $i \in \{O,L,U\}$, it denotes the query number needed for *O*rion, *L*ist decoding, or *U*nique decoding setting to guarantee security;
$\mathbb{F}$ is a field with a large multiplicative coset. $\mathbb{H}$ denotes a hash function. $n = 2^\mu$ denotes the polynomial size;
The PCS listed without assumptions relies solely on random oracle (RO) [28], which is plausibly post-quantum secure.

- **Implementation, applications and evaluations.** We have fully implemented our schemes and applied them to two practical cases: (i) constructing general zk-SNARKs [18, 45], and (ii) verifiable matrix multiplications [43]. We collected experimental data and demonstrated the competitive performance of our designs. Compared to Virgo [51], DeepFold achieves more than 3× faster prover time for a 26-variate multilinear polynomial. When compared to PolyFRIM [52], our proposal exhibits a 2× improvement in prover time, proof size, and verifier time. Furthermore, DeepFold outperforms BaseFold [48] by 3× in terms of proof size and verifier time, while maintaining similar prover time. When combining DeepFold with the HyperPlonk PIOP [18], the proof system is 3.6× faster than HyperPlonk + mKZG [35]. When integrated with Libra [45], the result is 2.5× faster than Libra + Virgo. Finally, we utilized our arbitrary-length input commitment to assist with verifiable matrix multiplication with dimensions 768 × 2304, an actual use case in GPT-2 [36]. Experiments showcase a 2.4× improvement in prover time compared to naive approaches.

1.2 Related Work

Multilinear PCS. In Table 1, we summarize the properties of related schemes for a comparison. Additionally, we list the concrete proof sizes for all schemes with a complexity of $O(s\log^2 n)$, where $\mu = 26$ and s is chosen differently by each scheme to ensure soundness. For all schemes based on RS code, we set the code rate to $\frac{1}{8}$, the same as the default setting in Plonky2 [2].

The (univariate) KZG scheme can be adapted to the multilinear case [35] with logarithmic verifier time and proof size. However, mKZG also inherits the issues of trusted setup and expensive group operations. Bulletproofs [16] does not require a trusted setup and relies solely on the Discrete Log (DLog) assumption. The problem is that verification takes $O(n)$ group exponentiations, making it often impractical for real-world applications. Hyrax [44] generalizes Bulletproofs by trading off proof size to improve verifier time, but an intrinsic limitation [31] is that the complexity product of the proof size and the verifier time cannot be smaller than $O(n)$. DARK [17] achieves logarithmic proof size and verifier time but relies on strong RSA and adaptive root assumptions. It is transparent only if class groups are used, which are known for slower group operations.

From another path, PCS based on error-correcting codes (ECC) relies solely on lightweight hash functions and is therefore transparent. Another advantage is that they avoid expensive group operations. There are two families of ECC-based PCS known for efficient prover performance and acceptable verifier time and proof size: (i) FRI-based schemes and (ii) tensor code-based schemes. The latter family of constructions is derived from the concepts introduced in Ligero [3] and has been recently enhanced in Brakedown [22] and Orion [47]. Without relying on proof recursion, Brakedown [22] achieves linear-time prover but suffers from a relatively slow verifier and a large proof size of $O(\lambda\sqrt{n})$, where λ is security parameter. On the other hand, Orion [47] reduces the proof size to $O(s_O\log^2 n)$ through proof composition with Virgo, where s_O represents the query number. However, s_O exceeds 1500, resulting in a large concrete proof size. In contrast, DeepFold and other FRI-based multilinear PCS [48, 51, 52] also yield a poly-logarithmic proof size and verifier time, but the concrete proof size is much smaller, as elaborated below.

FRI-based PCS. FRI-based PCS is originally designed for univariate polynomials [5]. The correctness relies on the fact that if $f(z) = y$ for a univariate polynomial $f(x)$ of degree n, then $\frac{f(x)-y}{x-z}$ is a polynomial of degree $(n-1)$. This quotient operation is inherently univariate, therefore obstructing the generalization of FRI to multivariate polynomials.

To overcome this limitation, Virgo [51] treats a multilinear polynomial evaluation as a univariate sumcheck protocol [8] and employs a GKR protocol [21] for FFT delegation, resulting in a prover overhead of $O(n \log n)$. PolyFRIM [52] adopts the idea of Gemini [14] by committing to additional $\log n$ polynomials to achieve $O(n)$ prover time. Similarly, the ZeromorphFRI [29] prover also commits to additional $\log n$ polynomials and batch-proves that all of them are of low degree. These two approaches run FRI nearly twice, concretely doubling the overhead for prover, verifier and proof size.

Recently, BaseFold [48] discovered that the last scalar sent by the prover in FRI is a random evaluation of the committed polynomial. It combines FRI with a multivariate sumcheck to achieve the most efficient FRI-based PCS. However, FRI is used in an unconventional way in [48], which restricts the scenario of BaseFold to only unique decoding settings, resulting in more query numbers and a larger proof size. This is precisely the problem we aim to address in this work.

FRI **query number.** The soundness of FRI relies on sufficient repetition of queries, the number of which is determined by the *distance preservation bound*. A larger bound allows for fewer queries, resulting in a smaller proof size. A series of studies [7,10,12] have been conducted to improve the distance preservation bound, thereby enhancing the efficiency of FRI.

In the original FRI paper [5], the bound is defined by the unique decoding radius $\frac{1-\rho}{2}$, where ρ is the code rate. Ben-Sasson et al. [12] successfully surpassed this unique decoding limit by pushing the bound to $J_\varepsilon(J_\varepsilon(\Delta V))$, which is approximately $1-\sqrt[4]{\rho}$ for RS codes. Later, Ben-Sasson et al. [10] further proposed the "one-and-a-half" Johnson function bound, increasing the bound to about $1-\sqrt[3]{\rho}$. Moreover, in the same paper, they introduced the *Domain Extending for Eliminating Pretenders* (DEEP) [10] technique, which effectively raises the bound to any *list decoding radius* by making lightweight adaptations to the protocol. This bound is theoretically optimal for FRI. Under a widely accepted conjecture on RS codes [11] (cf. Section 2.2), this bound could potentially be pushed to $1-\rho+\varepsilon$. In [7], Ben-Sasson et al. proved a bound of $1-\sqrt{\rho}$ for Reed-Solomon codes without using the DEEP technique. We will further elaborate on the distance preservation bound and the DEEP technique in subsequent sections.

Recently, Arnon et al. [4] proposed STIR to achieve fewer queries than DEEP-FRI, which can be used to construct univariate PCS. However, leveraging STIR to construct efficient multilinear PCS remains an open problem.

2 Preliminaries

Notation. We use λ to denote the security parameter. We use ε to denote any real number larger than 0. For $n \in \mathbb{N}$, let $[n]$ be the set $\{1,\cdots,n\}$. $\mathsf{negl}(\lambda)$ is used to denote a *negligible* function. A probability of $1-\mathsf{negl}(\lambda)$ is said to be *overwhelming*. We use $\mathbb{F}$ to denote a field such that $\log(|\mathbb{F}|) = \Omega(\lambda)$. For a vector $\vec{v} = (v_1,\cdots,v_n)$, let $\vec{v}_{[i:j]}$ denote the sub-vector $(v_i,\cdots,v_j)$, and let $\vec{v}_{[i:]}$ denote $(v_i,\cdots,v_n)$. For a function $f: A \to B$, and $D = \{d_1,\cdots,d_n\} \subseteq A$, let $f|_D$ denote the vector of evaluations $(f(d_1),\cdots,f(d_n))$. Let $\mathbb{F}^{\leq d}[X_1,\cdots,X_\mu]$ denote the set of all μ-variate polynomials with a degree bound d. We use $\otimes$ to denote a tensor product, and $\langle\cdot,\cdot\rangle$ to denote an inner product.

Merkle tree. Merkle tree is a vector commitment scheme with linear prover time and logarithmic verifier time and proof size, with respect to the vector size. A Merkle tree consists of three algorithms. $rt \leftarrow \mathsf{MT.Commit}(\vec{v})$ outputs the Merkle tree root rt for vector $\vec{v}$. $(\{v_i\}_{i\in I}, path) \leftarrow \mathsf{MT.Open}(I,\vec{v})$ takes as input the query location set I and outputs the values $\{v_i\}_{i\in I}$ and verification paths $path$ for these leaves. Verification is done via $\mathsf{MT.Verify}(rt, I, \{v_i\}_{i\in I}, path)$. We use collision-resistant and non-invertible hash functions to instantiate a Merkle tree.

Lemma 1 (Multilinear extension)**.** *A multilinear polynomial is a multivariate polynomial in which the degree of each variable is at most one. For every function $f: \{0,1\}^\mu \to \mathbb{F}$, there is a unique multilinear polynomial $\tilde{f} \in \mathbb{F}^{\leq 1}[X_1,\cdots,X_\mu]$ such that $\tilde{f}(\vec{b}) = f(\vec{b})$ for all $\vec{b} \in \{0,1\}^\mu$. We call $\tilde{f}$ the multilinear extension of f, and $\tilde{f}$ can be expressed as*

$$\tilde{f}(\vec{X}) = \sum_{\vec{b}\in\{0,1\}^\mu} f(\vec{b}) \cdot \tilde{eq}(\vec{b},\vec{X})$$

where $\tilde{eq}(\vec{b},\vec{X}) = \prod_{i=1}^{\mu}\left(\vec{b}[i]\vec{X}[i] + (1-\vec{b}[i])(1-\vec{X}[i])\right)$.

Lemma 2 (Schwartz-Zippel Lemma)**.** *Let $f \in \mathbb{F}[X_1,\cdots,X_\mu]$ be a non-zero polynomial of total degree d over field $\mathbb{F}$. Let S be any finite subset of $\mathbb{F}$, and let $r_1,\cdots,r_\mu$ be μ field elements selected independently and uniformly from S. Then*

$$\Pr\left[f(r_1,\cdots,r_\mu) = 0\right] \leq \frac{d}{|S|}$$

Lemma 3 (Twin Polynomials)**.** *For μ-variate multilinear polynomial $\tilde{f}(X_1,\cdots,X_\mu)$, we denote univariate polynomial $f(X) \in \mathbb{F}^{<2^\mu}[X]$ which shares the same coefficients with $\tilde{f}$. More precisely, there exists a coefficient vector $\vec{f}$ of 2^μ length such that: for each $(x_1,\cdots,x_\mu) \in \mathbb{F}^\mu$,*

$$\tilde{f}(x_1,\cdots,x_\mu) = \langle\vec{f},(1,x_1)\otimes\cdots\otimes(1,x_\mu)\rangle$$

and for each $x \in \mathbb{F}$, $f(x) = \langle\vec{f},(1,x,x^2,\cdots,x^{2^\mu-1})\rangle$. We call $\tilde{f}$ and f are twin polynomials of each other.

Interactive Argument of Knowledge. An interactive argument for an NP relation $\mathcal{R}$ is a triple of algorithms $(\mathcal{G},\mathcal{P},\mathcal{V})$. $\mathcal{G}$ is a setup algorithm. $\mathcal{P}$ tries to convince $\mathcal{V}$ that there exists a witness w such that $(x,w) \in \mathcal{R}$ for a statement x through a multi-round communication. To qualify as an interactive Argument of Knowledge (AoK), we require that w can be efficiently extractable by an extractor $\mathcal{E}$.

Definition 1 (Interactive Argument of Knowledge (AoK)). *$(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is an interactive argument of knowledge for an NP language $\mathcal{L}_{\mathcal{R}}$ if the following holds:*

- ***Completeness.*** *For every pp $\leftarrow \mathcal{G}(1^\lambda)$, every $x \in \mathcal{L}_{\mathcal{R}}$ and $(x,w) \in \mathcal{R}$, $\mathsf{Pr}[\langle \mathcal{P}(w), \mathcal{V}\rangle(\mathsf{pp}, x) = \mathsf{accept}] = 1$.*
- ***Argument of Knowledge.*** *For any PPT prover $\mathcal{P}^*$, $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ and x, $\exists$ PPT extractor $\mathcal{E}$ s.t. $\mathsf{Pr}[\langle \mathcal{P}^*, \mathcal{V}\rangle(\mathsf{pp}, x) = \mathsf{accept}, (x,w) \notin \mathcal{R} | w \leftarrow \mathcal{E}(\mathsf{pp}, x)] \le \mathsf{negl}(\lambda)$.*

We say an interactive AoK is **succinct** if the running time of $\mathcal{V}$ and the total communication cost between $\mathcal{P}$ and $\mathcal{V}$ are $\mathsf{poly}(\lambda, |x|, \log |w|)$.

2.1 Polynomial Commitment

Definition 2 (Multilinear Polynomial Commitment Scheme). *A multilinear polynomial commitment scheme* PC *consists of a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$*:*

- $\mathsf{Setup}(1^\lambda, \mu) \to \mathsf{pp}$. *It takes the security parameter λ and $\mu \in \mathbb{N}$ (i.e., the number of variables in a polynomial) and outputs the public parameter* pp.
- $\mathsf{Commit}(\mathsf{pp}, \tilde{f}) \to (\mathcal{C}, \mathcal{D})$. *It takes a μ-variate multilinear polynomial $\tilde{f}$ and outputs a commitment $\mathcal{C}$ along with some auxiliary message $\mathcal{D}$ (e.g., some randomness).*
- $\mathsf{OpenPoly}(\mathsf{pp}, \mathcal{C}, \tilde{f}, \mathcal{D}) \to b$. *It takes the commitment $\mathcal{C}$, the multilinear polynomial f, and the auxiliary message $\mathcal{D}$ to verify the commitment and outputs a bit $b \in \{0,1\}$.*
- $\mathsf{Eval}(\mathsf{pp}, \mathcal{C}, \vec{z}, y, \mu, \tilde{f}, \mathcal{D}) \to b$. *It is an interactive AoK between a PPT prover $\mathcal{P}$ and a verifier $\mathcal{V}$. It takes $\vec{z}$ as the evaluation point, y as the result, and outputs $b = \langle \mathcal{P}(\tilde{f}, \mathcal{D}), \mathcal{V}\rangle(\mathsf{pp}, \mathcal{C}, \vec{z}, y, \mu)$. The algorithm only outputs $b = 1$ if it satisfies $\tilde{f}(\vec{z}) = y$.*

The scheme should satisfy *Completeness*, *Binding*, and *(Knowledge) Soundness*. The formal definition is presented in Appendix A.

Batch evaluations. This work considers a special case where the prover needs to evaluate multiple polynomials (possibly with different numbers of variables) at different points, which is referred to as *batch evaluation*. More precisely, let $\{\mu_i, \tilde{f}_i, \mathcal{C}_i\}_{i=1}^n$ represent n different (μ_i-variate) multilinear polynomials along with their commitments. An algorithm $\mathsf{BatchEval}(\mathsf{pp}, \{\mathcal{C}_i, \vec{z}_i, y_i, \mu_i, \tilde{f}_i, \mathcal{D}_i\}_{i=1}^n) \to b$ is an interactive AoK if it outputs $b = 1$ only when $\{\tilde{f}_i(\vec{z}_i) = y_i\}_{i=1}^n$ is satisfied.

2.2 Linear Codes and Reed-Solomon Code

Distance. For two vectors $\vec{u}, \vec{v} \in \mathbb{F}^n$, we use $\mathsf{Ham}(\vec{u}, \vec{v}) := \mathsf{Pr}_{i \in [n]}(\vec{u}[i] \neq \vec{v}[i])$ to denote their relative Hamming distance. For vector $\vec{u}$ and a vector set V, we use $\mathsf{Ham}(\vec{u}, V) := \min_{\vec{v} \in V} \{\mathsf{Ham}(\vec{u}, \vec{v})\}$ to denote their distance. When $\mathsf{Ham}(\vec{u}, \vec{v}) \le \delta$, we call $\vec{u}$ is δ-close with $\vec{v}$. When $\mathsf{Ham}(\vec{u}, \vec{v}) \ge \delta$, we call $\vec{u}$ is δ-far from $\vec{v}$.

Linear codes. Linear codes are a type of error-correcting codes, where the codewords form a linear-space.

Definition 3 (Linear Code). *An (n,k,δ)-linear error-correcting code C is a linear subspace of $\mathbb{F}^n$ defined by an injective mapping $E: \mathbb{F}^k \to \mathbb{F}^n$, where $E(\mathbb{F}^k)$ forms the linear subspace C and $\mathsf{Ham}(u,v)$ of different vectors $u, v \in C$ is at least δ. A codeword is a vector in this linear subspace. The code rate of a (n,k,δ)-linear code is $\rho = \frac{k}{n}$.*

Reed-Solomon codes. RS codes are a type of linear code where messages are encoded by a polynomial of degree at most $k-1$, determined by the k entries of the message. The polynomial is then evaluated at n points to obtain the codeword.

Definition 4 (RS Code). *A $[\mathbb{F}, L, \rho]$ RS Code is defined as follows:*

$$\mathsf{RS}[\mathbb{F}, L, \rho] = \{p|_L : p \in \mathbb{F}[X], \deg(p) \le \rho |L|\}$$

It is usually denoted as $\mathsf{RS}[L, \rho]$ when there is no ambiguity. In practice and in FRI*, L is usually a multiplicative subgroup of $\mathbb{F}$ with a size that is a power of 2.*

Definition 5 (List Size and List Decodability for RS Codes). *For $\vec{u} \in \mathbb{F}^n$, a linear code $V \subset \mathbb{F}^n$, and distance parameter $\delta \in [0,1]$, let $\mathsf{List}(\vec{u}, V, \delta) = B(\vec{u}, \delta) \cap V$, where $B(\vec{u}, \delta)$ is the Hamming ball of relative radius δ centered at $\vec{u}$. The code V is said to be (δ, L)-list-decodable if $|\mathsf{List}(\vec{u}, V, \delta)| \le L$ for all $\vec{u} \in \mathbb{F}^n$.*

Definition 6 (Unique and List Decoding Radius). *δ such that V is $(\delta, 1)$-list-decodable is called the unique decoding radius. δ such that V is $(\delta, \mathsf{poly}(|n|))$-list-decodable is called the list decoding radius.*

According to the known limitation [11], there is an optimistic and widely applied conjecture regarding the list decodability of RS codes [10], on which our PCS will be based. Note that many RS code-based PCS, like DEEP-FRI [10] and RedShift [27], all rely on this conjecture.

Conjecture 1 (List Decodability for RS Codes). *For every $\rho > 0$, there is a constant C_ρ such that for every* RS *code V, whose codewords are of length n and rate ρ, there has*

$$\mathcal{L}(\mathbb{F}, L, d = \rho|L|, 1 - \rho - \varepsilon) \le \left(\frac{|L|}{\varepsilon}\right)^{C_\rho}$$

where L is the evaluation domain of RS *code.*

We summarize the relationship between δ and the decodability of RS codes in Figure 1. In other words, when δ is a unique decoding radius ($\delta \le \frac{1-\rho}{2}$), for any $\vec{v}$, there is at most

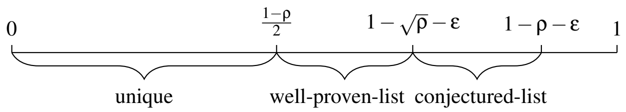

Figure 1: Illustration of RS code decodability.

one codeword that is δ-close to $\vec{v}$. When δ is a list decoding radius ($\delta \in (\frac{1-\rho}{2}, 1-\rho)$, where $[1-\sqrt{\rho}, 1-\rho)$ is conjectured), for any $\vec{v}$, there might be more than one (but not too many) codewords that are δ-close to $\vec{v}$.

Correlated agreements. The *correlated agreement* principle [7] (Figure 2) states that if a random linear combination of two given vectors u, u^* is δ-close ($\delta \le \Delta^*$, where $\Delta^* \in [0,1]$ is called the *distance preservation bound*) to a RS code with non-negligible probability, then each of the two vectors is δ-close to some codeword (v, v^* respectively) on *the same set of locations*. The advancement of the distance preservation bound has undergone extensive research, as discussed in Section 1.2, where the optimal bound $\Delta^* = 1-\rho-\varepsilon$ is derived from DEEP-FRI [10].

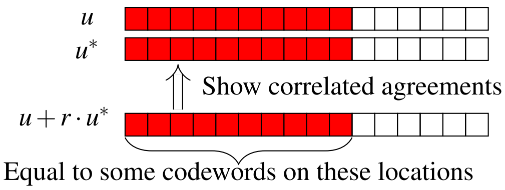
u
u^*
Show correlated agreements
$u + r \cdot u^*$
Equal to some codewords on these locations

Figure 2: Illustration of correlated agreement principle.

3 Technical Overview

Our main contribution in this paper is the reduction of the number of queries in BaseFold [48], resulting in a significantly smaller proof size. We first provide a quick review of BaseFold and the FRI technique behind it (and our construction), followed by an outline of challenges we address in this work and the core techniques we introduce.

3.1 Warmup: FRI and BaseFold

Recall that an RS-IOPP is designed to convince a verifier that a given vector $\vec{v}$ is close to some codeword, where the verifier is given only oracle access to $\vec{v}$. FRI is a specifically designed RS-IOPP, enabling the prover $\mathcal{P}$ to convince the verifier $\mathcal{V}$ that the committed vector $\vec{v}$ is Δ-close to $\mathsf{RS}[\mathbb{F}, L_0, \rho]$, where L_0 is a multiplicative subgroup of $\mathbb{F}$. To avoid confusion, here we use $\Delta \in [0,1]$ to denote the maximum acceptable relative Hamming distance in the FRI protocol, while $\delta = \mathsf{Ham}(\vec{v}, \mathsf{RS}[\mathbb{F}, L_0, \rho])$ denotes the actual distance between $\vec{v}$ and the RS code.

The FRI protocol. For any honest prover, $\vec{v}$ should equal $f^{(0)}|_{L_0} \in \mathsf{RS}[\mathbb{F}, L_0, \rho]$, where $f(X) = f^{(0)}(X)$ denotes a univariate polynomial whose degree is less than $2^\mu := \rho|L_0|$. Define $L_{i+1} = \{x^2 : x \in L_i\}$. The FRI protocol proceeds in two phases: the commit phase and the query phase. We provide an overview of FRI protocol below.

Commit phase. The commit phase consists of μ rounds. In the i-th round, $\mathcal{P}$ folds the polynomial $f^{(i-1)}$ into $f^{(i)}$, reducing its length by half. Specifically, $f^{(i-1)}(X)$ can be expressed as a combination of two half-degree polynomials, $f_E^{(i)}$ and $f_O^{(i)}$:

$$f^{(i-1)}(X) = f_E^{(i)}(X^2) + X \cdot f_O^{(i)}(X^2) \tag{1}$$

Given a random challenge $r_i \in \mathbb{F}$ from $\mathcal{V}$, a new polynomial $f^{(i)}$ is computed as a random linear combination of $f_E^{(i)}, f_O^{(i)}$:

$$f^{(i)}(X) = f_E^{(i)}(X) + r_i \cdot f_O^{(i)}(X) \tag{2}$$

$\mathcal{P}$ then sends a Merkle commitment of the vector $\vec{v}^{(i)} = f^{(i)}|_{L_i}$ to $\mathcal{V}$. Since the degree is halved in each round, in the final round, $f^{(\mu)}$ should be a scalar if $\mathcal{P}$ is honest.

Query phase. At the end of the commit phase, $\mathcal{V}$ queries each of the μ committed vectors at random locations. These openings are used to verify the consistency of the folding performed by $\mathcal{P}$ in each round during the commit phase. To maintain the soundness of the protocol, the query process should be repeated multiple times, denoted by the query repetition count s.

The soundness of the FRI protocol can be derived from Theorem 1 [7]. Intuitively, the correlated agreement principle (Section 2.2) ensures that the random linear combination (Equation 2) does not decrease the distance between $\vec{v}^{(i)}$ and its corresponding RS code when reducing $\vec{v}^{(i-1)}$ to $\vec{v}^{(i)}$. Therefore, the folding steps between $f^{(i-1)}$ and $f^{(i)}$ are valid, making the protocol result acceptable for $\mathcal{V}$. Moreover, according to Theorem 1, to achieve λ-bit security, the query repetition count s must satisfy $(1-\Delta)^s < 2^{-\lambda}$ if $\Delta < \Delta^*$.

Theorem 1. *For any $\Delta < \Delta^*$, where Δ^* is the distance preservation bound, if $\delta = \mathsf{Ham}(\vec{v}, \mathsf{RS}[\mathbb{F}, L_0, \rho]) \ge \Delta$, with overwhelming probability, after the commit phase, at least Δ-proportional locations on L_0 allows $\mathcal{V}$ to find an error.*

Recap of BaseFold. Let $\tilde{f}(X_1, \ldots, X_\mu)$ denote a μ-variate multilinear polynomial, which is the twin polynomial of the univariate polynomial f. BaseFold [48] makes a key observation that for an honest prover, the last round polynomial $f^{(\mu)}$ in FRI is actually a scalar equal to $\tilde{f}(r_1, \ldots, r_\mu)$, where $r_1, \ldots, r_\mu$ are the random challenges from $\mathcal{V}$. Moreover, [48] proves this property holds even for malicious provers when Δ is a *unique decoding radius* (cf. Section 2.2).

Leveraging this observation, [48] provides an efficient construction of multilinear PCS. The commitment of $\tilde{f}$ is set as the Merkle commitment of the vector $\vec{v} = f^{(0)}|_{L_0}$. To validate the evaluation of $\tilde{f}(\vec{z})$, where $\vec{z}$ is a given evaluation point, [48] introduces an elegant approach that runs an FRI protocol and a classic sumcheck protocol [34] in parallel. Specifically, $\mathcal{P}$ and $\mathcal{V}$ run both a μ-round FRI protocol and a sumcheck protocol simultaneously, sharing the same random challenges sampled by $\mathcal{V}$. The intuition underlying this approach is: (i) The sumcheck protocol ensures the correctness of $\tilde{f}(\vec{z}) = \sum_{\vec{b}\in\{0,1\}^{\mu}} \tilde{f}(\vec{b}) \cdot \tilde{eq}(\vec{b},\vec{z})$, reducing the correctness of $\tilde{f}(\vec{z})$ to a random evaluation $\tilde{f}(\vec{r})$, and (ii) With the aforementioned observation, the FRI protocol on f should precisely provide a correct $\tilde{f}(\vec{r})$.

Limitation of excessive queries. In [48], one major issue we identify is the relatively large proof size, which is due to the chosen large s in the FRI protocol. To understand why such a large s is needed in BaseFold, recall that to achieve λ-bit security for the soundness of FRI, the query repetition count s must satisfy $(1-\Delta)^s < 2^{-\lambda}$ (cf. Theorem 1). Since the proof size is proportional to s, it is crucial to seek a larger Δ to reduce the proof size. Because [48] operates *only* in the unique decoding radius setting, the largest Δ can be chosen is $\frac{1-\rho}{2}$ (Section 2.2). Concretely, if $\rho = \frac{1}{8}$, then to achieve 100-bit security, the query count must exceed 120, which translates to a proof size of over 600 KB.

3.2 Overview of DeepFold

DeepFold aims to reduce s in BaseFold, thereby decreasing the proof size. One may consider the most straightforward approach is to transition directly to the *list decoding radius* setting, where the limit of Δ can be pushed to $1-\rho-\varepsilon$ under Conjecture 1.

Problem of transition. However, we argue that this direct transition can not be *simply* applied to BaseFold. Recall that in BaseFold, FRI is actually used in a "specific" manner: instead of a Reed-Solomon IOPP, FRI is further used to provide the correctness of $f^{(\mu)}$. Originally, FRI only requires that the given vector $\vec{v}$ is close to some RS code, without specifying in the i-th round *which codeword* the committed vector $\vec{v}^{(i)}$ should be close to. Therefore, $\vec{v}^{(i)}$ could be Δ-close to multiple RS codewords, including those irrelevant to the original vector $\vec{v}$. This "gap" precisely implies why BaseFold works smoothly *only* under the unique radius setting: for Equation 2, in the i-th round, there is at most one $f^{(i)}$ that satisfies the distance requirement (Def. 6), and thus, in the end, $f^{(\mu)}$ is uniquely determined, even for a malicious prover.

Yet, when transitioning to the list decoding radius setting where $\Delta > \frac{1-\rho}{2}$, multiple codewords are decodable from a given vector (Def. 6). Therefore, for Equation 2, in each round, a malicious prover may have the potential to derive a different $f^{(i)\prime}$ to pass the check. As a result, the final scalar $f^{(\mu)\prime}$ could be an incorrect value. In Figure 3, we provide an illustration of this problem. A malicious prover may rely on the random combination of some "garbage" values (the blue and yellow positions) to derive an incorrect proximity while still satisfying the Δ-close correlated agreements.

We found that there is actually no security guarantee against this type of attack. Therefore, it remains a technical problem to ensure the correctness of $f^{(\mu)}$ for a malicious prover: we must develop a method to make sure that in the list decoding radius setting, the vector $\vec{v}^{(i)}$ committed in the i-th round is *not* Δ-close to any codeword other than $f^{(i)}$, where $f^{(i)}$ is the twin polynomial of $\tilde{f}(r_1,\cdots,r_i,X_{i+1},\cdots,X_{\mu})$.

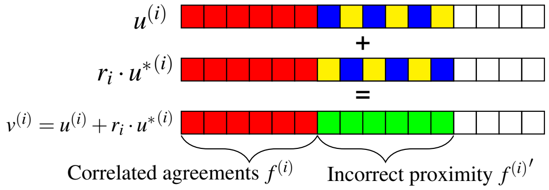
$u^{(i)}$
+
$r_i \cdot u^{*(i)}$
=
$v^{(i)} = u^{(i)} + r_i \cdot u^{*(i)}$
Correlated agreements $f^{(i)}$
Incorrect proximity $f^{(i)\prime}$

Figure 3: Illustration of the problem when transitioning to the list decoding setting: In the i-th round, the vector $\vec{v}^{(i)}$ may be close to multiple codewords.

Idea 1: Using DEEP for transition. Our idea is that in each i-th round of FRI, let $\mathcal{V}$ query another evaluation outside L_i, which could help in binding the unique polynomial $f^{(i)}$. This idea is inspired by the *Domain Extending for Eliminating Pretenders* (DEEP) technique introduced in DEEP-FRI [10]. Although not originally intended for this purpose, we found the DEEP technique can be adapted to solve our problem.

Specifically, in each i-th round, $\mathcal{V}$ makes two additional random queries, $f^{(i-1)}(\pm\alpha_i)$ (which implies $f^{(i)}(\alpha_i^2)$), where α_i is a challenge from $\mathcal{V}$, just as in [10]. The intuition behind this is: In the list decoding radius setting, there may be more than one but *not too many* codewords within the radius Δ. Since α_i is chosen from the entire field $\mathbb{F}$, with overwhelming probability, there would not be two polynomials in $\mathsf{List}(\vec{v}^{(i)},\mathsf{RS},\Delta)$ that evaluate the same on α_i^2. Thus, $f^{(i)}(\alpha_i^2)$ is sufficient to bind the vector to the unique polynomial $f^{(i)} \in \mathsf{List}(\vec{v}^{(i)},\mathsf{RS},\Delta)$, thereby avoiding the aforementioned problem. To ensure the correctness of this approach, we provided a security proof in Theorem 4.

Idea 2: Using DEEP without quotient. Now, the challenge comes in enabling $\mathcal{V}$ to validate the correctness of $f^{(i)}(\alpha_i^2)$ in each i-th round, which remains a non-trivial technical problem. We note that in DEEP-FRI [10], the authors also attempted to address a similar issue. Since their target is a univariate PCS, the authors combine the validation of $f^{(i)}(\alpha_i^2)$ with the original folding formula (Equation 2). Specifically, in the i-th round of the FRI commit phase, the folding from $f^{(i-1)}$ to $f^{(i)}$ is adapted according to a new formula:

$$f^{(i)}(X) = \frac{(f_E^{(i)}(X) + r_i \cdot f_O^{(i)}(X)) - (f_E^{(i)}(\alpha_i^2) + r_i \cdot f_O^{(i)}(\alpha_i^2))}{X - \alpha_i^2} \quad (3)$$

where r_i is the original round challenge and α_i is the extra query point. In this way, the validation of each $f^{(i)}(\alpha_{i-1}^2)$ is

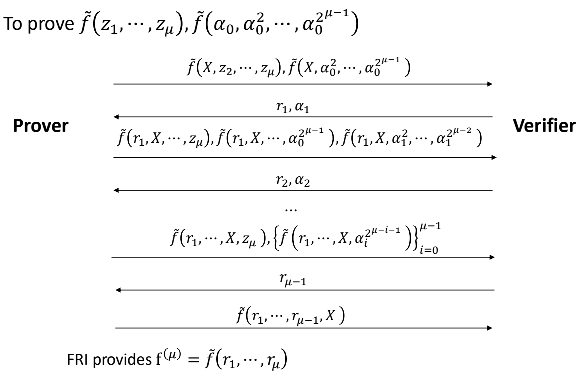
To prove $\tilde{f}(z_1,\cdots,z_\mu), \tilde{f}(\alpha_0,\alpha_0^2,\cdots,\alpha_0^{2^{\mu-1}})$
Prover
Verifier
$\tilde{f}(X,z_2,\cdots,z_\mu), \tilde{f}(X,\alpha_0^2,\cdots,\alpha_0^{2^{\mu-1}})$
r_1,α_1
$\tilde{f}(r_1,X,\cdots,z_\mu), \tilde{f}(r_1,X,\cdots,\alpha_0^{2^{\mu-1}}), \tilde{f}(r_1,X,\alpha_1^2,\cdots,\alpha_1^{2^{\mu-2}})$
r_2,α_2
...
$\tilde{f}(r_1,\cdots,X,z_\mu), \{\tilde{f}(r_1,\cdots,X,\alpha_i^{2^{\mu-i-1}})\}_{i=0}^{\mu-1}$
$r_{\mu-1}$
$\tilde{f}(r_1,\cdots,r_{\mu-1},X)$
FRI provides $f^{(\mu)} = \tilde{f}(r_1,\cdots,r_\mu)$

Figure 4: Visualization of Deepfold evaluation.

cleverly reduced to the IOPP of $f^{(i)}$ in the following rounds. This technique is known as the *quotient* in [10].

Nonetheless, we find that this technique is not applicable in our context. The reason is that the folding from Equation 3 hinders the aforementioned important property, as the last scalar in FRI, $f^{(\mu)}$, no longer equals $\tilde{f}(\vec{r})$. This makes the construction of a multilinear PCS extremely challenging.

Therefore, another technical contribution of our work is a new method to ensure the correctness of these extra evaluations on $\{\alpha_i\}$. From a high-level perspective, instead of making quotients like in [10], in DeepFold, the evaluations $\{f(\alpha_i^2)\}$ are *deeply folded*. Suppose in the i-th round, $\mathcal{V}$ needs to check the correctness of $f^{(i-1)}(\pm\alpha_i)$. Let $\tilde{f}^{(i)} := \tilde{f}(r_1,\cdots,r_i,X_{i+1},\cdots,X_\mu)$ denote the twin of $f^{(i)}$. Leveraging the relationship between twin polynomials, we derive that $f^{(i-1)}(\pm\alpha_i) = \tilde{f}^{(i-1)}(\pm\alpha_i,\alpha_i^2,\ldots,\alpha_i^{2^{\mu-i}})$, from which $\mathcal{V}$ can extrapolate $f^{(i)}(\alpha_i^2) = \tilde{f}^{(i)}(\alpha_i^2,\ldots,\alpha_i^{2^{\mu-i}})$. In the j-th round ($j > i$), $\mathcal{P}$ additionally sends $f^{(j-1)}(-\alpha_i^{2^{j-i}})$, after which the correctness of the $f^{(i-1)}(\pm\alpha_i)$ can be reduced to $f^{(j)}(\alpha_i^{2^{j-i+1}})$. Finally, all the checks on $f^{(i-1)}(\pm\alpha_i)$ converge to the same point $f^{(\mu)} = \tilde{f}(r_1,\cdots,r_\mu)$, which is precisely provided by FRI. Furthermore, in the Eval procedure of DeepFold, the evaluation $\tilde{f}(\vec{z})$ can also be validated in a similar manner. We visualize the whole procedure in Fig. 4.

Until now, we have successfully made the transition to the list decoding radius setting, where the maximum distance requirement Δ can be set to approximately $1-\rho-\varepsilon$, and the query repetition count s is reduced to around 34 times. Our experiments suggest that this improves the proof size of BaseFold by 3× under the same conditions as before.

Arbitrary input length via batching. We introduce a technique to help with committing and evaluating multilinear polynomials encoded from inputs of arbitrary length. This can be achieved from a batched DeepFold for multiple polynomials of different sizes. For example, for a vector $\vec{v}$ of length $2^{2n}+2^n$, we need to encode $\vec{v}$ as a multilinear polynomial $\tilde{f}$ with $2n+1$ variables. However, a naive approach to handling $\tilde{f}$ would result in doubled prover overhead. Instead, we treat the inputs as two multilinear polynomials, $\tilde{f}_1$ and $\tilde{f}_2$, with $2n$ and n variables, respectively, such that:

$$\tilde{f}(X_0,\cdots,X_{2n}) = (1-X_0)\cdot\tilde{f}_1(X_1,\cdots,X_{2n}) + X_0\cdot\prod_{i=1}^{n}(1-X_i)\cdot\tilde{f}_2(X_{n+1},\cdots,X_{2n})$$

One way to evaluate $\tilde{f}(z_0,\ldots,z_{2n})$ is for $\mathcal{P}$ to simultaneously open $y_1 = \tilde{f}_1(z_1,\ldots,z_{2n})$ and $y_2 = \tilde{f}_2(z_{n+1},\ldots,z_{2n})$, but this would require two proofs, nearly doubling the proof size.

To address this, we present an efficient batch evaluation algorithm building upon the techniques applied in DeepFold. When proving y_1 and y_2, the idea is to first reduce the evaluation of y_1 to $\tilde{f}_1(r_1,\ldots,r_n,z_{n+1},\ldots,z_{2n})$ through n rounds of interactions. Then, in the last n rounds, the prover incorporates $\tilde{f}_1(r_1,\ldots,r_n,z_{n+1},\ldots,z_{2n})$ and $\tilde{f}_2(z_{n+1},\ldots,z_{2n})$ through random linear combinations, allowing the validation to be done simultaneously. As a result, the batch evaluation protocol achieves optimal prover complexity, which is linearly proportional to the total length of all polynomials. Furthermore, the proof size and verifier time remain almost the same as the cost of evaluating $\tilde{f}_1$ independently. This idea can be extended to accommodate more polynomials, even when the evaluation points differ for each polynomial. Finally, we leverage the batching technique to construct a zero-knowledge version of DeepFold, which is also an interesting contribution, as achieving zero-knowledge for FRI-based PCS remains non-trivial.

4 DeepFold: Multilinear Polynomial Commitment based on Reed-Solomon Code

In this section, we present a novel multilinear PCS, which we call DeepFold. The high-level idea is to expand the proximity bound to the entire list decoding radius setting, thereby substantially reducing the number of queries needed and improving the proof size of BaseFold [48].

4.1 DeepFold Multilinear PCS

Let $\tilde{f}$ denote the μ-variate multilinear polynomial, and let $f = f^{(0)}$ denote its univariate twin polynomial. As discussed in the overview, DeepFold also utilizes FRI as a subroutine to provide $f^{(\mu)}$ and thereby validate the correctness of $\tilde{f}(\vec{r})$, where $\vec{r}$ is the round challenges in the protocol. The transition to list decoding radius setting reduces proof size but introduces several challenges, which we summarize as follows:

1. The PCS may lose its binding property as there could be multiple codewords Δ-close to the committed vector $\vec{v}$.
2. In each i-th round of the FRI protocol, there could be multiple codewords close to the vector $\vec{v}^{(i)}$. A malicious prover may choose the codeword arbitrarily, thereby providing an incorrect $f^{(\mu)}$.

3. The quotient technique from [10] cannot be applied in our context, as it ruins the nice structure of twin polynomials.

We claim that DeepFold addresses the above challenges. Below, we first present the protocol:

- $\mathsf{Setup}(1^\lambda, \mu) \to \mathsf{pp}$. It takes the security parameter λ and the number of variables μ, and outputs the public parameter $\mathsf{pp} = \{\mathbb{F}, L_0, s\}$, where s is the query repetition number satisfying $(1-\Delta)^s \leq 2^{-\lambda}$. In our setting, Δ can be set as $1-\rho-\varepsilon$, where ρ is the code rate.
- $\mathsf{Commit}(\mathsf{pp}, \tilde{f}) \to (\mathcal{C}, \mathcal{D})$. It takes pp and $\tilde{f}$, and outputs the commitment $\mathcal{C} = \langle rt_0, \alpha, c\rangle$ according to the following:

```
1: Let $\vec{v} = f^{(0)}|_{L_0}$, $\mathcal{P}$ sends $\mathsf{MT.Commit}(\vec{v}) \to rt_0$ to $\mathcal{V}$.
2: $\mathcal{V}$ sends $\alpha \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
3: $\mathcal{P}$ computes $c := f^{(0)}(\alpha)$ and sends $c$ to $\mathcal{V}$.
```

- $\mathsf{OpenPoly}(\mathsf{pp}, \mathcal{C}, \tilde{f}, \mathcal{D}) \to b$: $\mathcal{P}$ opens the commitment $\mathcal{C}$ and sends $\vec{v}$ to $\mathcal{V}$. $\mathcal{V}$ then checks (i) the Merkle root equals the committed root by reconstructing the Merkle tree, (ii) $f^{(0)}(\alpha) = c$, and (iii) $\mathsf{Ham}(\vec{v}, f^{(0)}|_{L_0}) < \Delta$. Note that although $\vec{v} = f^{(0)}|_{L_0}$ must hold for the honest prover, we allow a discrepancy between them in this definition to ensure the soundness of our PCS, as defined in Appendix A.
- $\mathsf{Eval}(\mathsf{pp}, \mathcal{C}, \vec{z}, y, \mu; \tilde{f}, \mathcal{D}) \to b$. Given the evaluation point $\vec{z} = (z_1, \cdots, z_\mu)$, $y = \tilde{f}(\vec{z})$, $\tilde{f}$, $\mathcal{P}$ and $\mathcal{V}$ run the following:

```
1: Let $A_0 := \{\vec{z}, \vec{\alpha}\}$, and $\vec{\alpha} = (\alpha^{2^0}, \alpha^{2^1}, \cdots, \alpha^{2^{\mu-1}})$.
2: For each round $i$, where $i \in [\mu]$,
    a. $\mathcal{V}$ sends $\alpha_i \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$. Let $A_{i-1} := \{A_{i-1}, \vec{\alpha}_i\}$, where $\vec{\alpha}_i = (\alpha_i^{2^0}, \alpha_i^{2^1}, \cdots, \alpha_i^{2^{\mu-i}})$.
    b. If $i = \mu$, $\mathcal{P}$ sends a linear function $g(X) := \tilde{f}(r_1, \ldots, r_{i-1}, X)$ to $\mathcal{V}$. Otherwise, let $A_i := \emptyset$, for each $\vec{w} \in A_{i-1}$, $\mathcal{P}$ sends $g_{\vec{w}_{[2:]}}(X) := \tilde{f}(r_1, \ldots, r_{i-1}, X, \vec{w}_{[2:]})$ to $\mathcal{V}$. Append $\vec{w}_{[2:]}$ into $A_i$.
    c. $\mathcal{V}$ sends $r_i \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
    d. $\mathcal{P}$ computes $f^{(i)}(X) = f_E^{(i)}(X) + r_i \cdot f_O^{(i)}(X)$, where $f_E^{(i)}(X), f_O^{(i)}(X)$ satisfy that:
           $$f^{(i-1)}(X) = f_E^{(i)}(X^2) + X \cdot f_O^{(i)}(X^2)$$
    e. Let $\vec{v}^{(i)} = f^{(i)}|_{L_i}$. If $i = \mu$, $\mathcal{P}$ sends a scalar $f^{(\mu)} \in \mathbb{F}$ to $\mathcal{V}$. Otherwise, $\mathcal{P}$ sends $\mathsf{MT.Commit}(\vec{v}^{(i)}) \to rt_i$.
3: $\mathcal{V}$ checks $g_{\vec{z}_{[2:]}}(z_1) = y$, $g_{\vec{\alpha}_{[2:]}}(\alpha) = c$, $g(r_\mu) = f^{(\mu)}$. For each round $i$, where $i \in [\mu]$,
    a. For each $\vec{w} \in A_{i-1}$, if $i < \mu$, $\mathcal{V}$ checks $g_{\vec{w}}(r_i) = g_{\vec{w}_{[2:]}}(w_1)$; otherwise, $\mathcal{V}$ checks $g_{\vec{w}}(r_i) = g(w_1)$.
4: Repeat the following query process $s$ times:
    a. $\mathcal{V}$ sends $\beta_0 \xleftarrow{\$} L_0$ to $\mathcal{P}$. For $i \in [\mu]$, define $\beta_i := \beta_{i-1}^2$.
    b. For $i \in [\mu]$, $\mathcal{P}$ opens $f^{(i-1)}(\beta_{i-1})$ and $f^{(i-1)}(-\beta_{i-1})$ using $\mathsf{MT.Open}$.
    c. $\mathcal{V}$ checks the results using $\mathsf{MT.Verify}$.
    d. For $i \in [\mu]$, $\mathcal{V}$ checks the following triple is on a common line.
           $$\left(\beta_{i-1}, f^{(i-1)}(\beta_{i-1})\right), \left(-\beta_{i-1}, f^{(i-1)}(-\beta_{i-1})\right), \left(r_i, f^{(i)}(\beta_i)\right)$$
5: If all above checks pass, $\mathcal{V}$ outputs 1; 0 otherwise.
```

The commit and evaluation algorithms can be made non-interactive through a Fiat-Shamir heuristic transformation, as the protocol is essentially public-coin. Now, we provide some discussions on the critical steps to address the aforementioned challenges. First, to ensure binding to a unique polynomial, we add a random query $f^{(0)}(\alpha)$ into the commitment. As demonstrated in Theorem 4, the extra evaluation $f^{(0)}(\alpha) = c$ is sufficient to maintain the binding property. Similarly, we add one more evaluation $f^{(i)}(\alpha_i)$ in each round to ensure that only one codeword is decodable from the committed vector, which ensures the correctness of $f^{(\mu)}$. Finally, we propose a new method to validate the above evaluations: For each evaluation $\tilde{z}$ and $\{f^{(i)}(\alpha_i)\}$, the interactions in Step 2-b and Step 3-a reduce the check to the next round, finally converging at $\tilde{f}(r_1, \ldots, r_\mu)$. Since the value is provided by $\tilde{f}^{(\mu)}$, this concludes the check. We note that DeepFold doesn't explicitly invoke a sumcheck as what BaseFold does. Instead, we use the same technique of folding DEEP evaluations to fold the original claim in Step b.

Complexity analysis. We present the complexity analysis for DeepFold. For simplicity, we consider the code rate ρ as constant, therefore $O(L_0) = O(n)$.

Commit. During the commitment, $\mathcal{P}$ is initially tasked with calculating the RS code of the polynomial coefficient vector, which requires an FFT, leading to a time complexity of $O(n \log n)$ field operations. Besides this, $\mathcal{P}$ computes $O(n)$ hashes to obtain the Merkle commitment.

Evaluate. In the i-th round, $\mathcal{P}$ computes $f^{(i)}|_{L_i}$ and its Merkle tree from $f^{(i-1)}|_{L_{i-1}}$ with a time complexity of $O(L_i)$, so the total complexity to compute all Merkle commitments is also $O(n)$. $\mathcal{P}$ also needs to compute $i + O(1)$ evaluations on each $\tilde{f}^{(i)}$. Each evaluation requires $O(2^{\mu-i})$ time. So the total time complexity to compute all these evaluations is $\sum_{i=0}^{\mu-1}(i + O(1)) \times O(2^{\mu-i}) = O(2^\mu) = O(n)$. Thus, the total prover overhead is $O(n)$ field operations and $O(n)$ hashes.

Proof size. In addition to the Merkle paths equivalent to traditional FRI, which include a proof size of $O(s \cdot \log^2 n)$, there are an additional $i + O(1)$ field elements to be sent in the i-th round, so the total communication overhead is $\sum_{i=0}^{\mu-1}(i + O(1)) = O(\mu^2) = O(\log^2 n)$. Thus, the total proof size is $O(s \cdot \log^2 n)$.

Verify. Similar with the analysis of proof size, the verifier complexity includes a traditional FRI verification plus a $O(\log^2 n)$ overhead, so the total verifier complexity is $O(s \cdot \log^2 n)$.

Comparison with BaseFold. The prover asymptotic complexity of DeepFold is identical to that of BaseFold. The query number s should satisfy $(1-\Delta)^s < 2^{-\lambda}$ (Theorem 1), where Δ is the maximum acceptable distance of the FRI protocol. DeepFold operates under the list decoding radius setting, where Δ can be set at approximately $1-\rho-\varepsilon$, while BaseFold only works under the unique decoding radius setting, where Δ can at most be set at $\frac{1-\rho}{2}$. Therefore, the query repetition number s of DeepFold is smaller. The practical performance results in Section 7 show that our scheme substantially reduces proof size and verifier time.

Security analysis. Due to the space limitation, we provide the analysis of completeness, binding, and soundness for DeepFold in Appendix B.

5 Batching Evaluation on Different-size Polynomials

A traditional method for batch evaluation of univariate polynomials involves $\mathcal{P}$ randomly combining all polynomials and then evaluating the combined result. However, it remains unknown how to apply this technique to multilinear polynomials of different sizes. A naive approach is for $\mathcal{P}$ to pad the shorter polynomials to match the largest one, which is costly. To address this, in this section, we introduce a technique to efficiently batch evaluate multiple multilinear polynomials of different sizes without any overhead from padding.

5.1 Batch Evaluation with DeepFold

The key insight is that in the evaluation of DeepFold, a multilinear polynomial is split and randomly folded into a smaller one, as described in Step 2-d. Thus, when handling multiple multilinear polynomials of different sizes, we can intuitively combine them randomly when the larger polynomial is folded to the same size as the smaller one. Specifically, for example, when batch evaluating two multilinear polynomials, $\tilde{f}_0$ with μ variates and $\tilde{f}_1$ with ℓ variates ($\ell < \mu$), $\tilde{f}_0$ is first folded into $f_0^{(\mu-\ell)}$ in the $(\mu-\ell)$-th round. Now the two polynomials can be incorporated together since they have the same size. In all subsequent i-th rounds, we handle the univariate polynomial $f_0^{(i)} + \gamma_i \cdot f_1^{(i)}$, which effectively reduces the overhead compared to separately dealing with each polynomial.

Now, we present DeepFold.BatchEval, which batch evaluates μ multilinear polynomials, each with a different number of variables. Note that the polynomials are evaluated at the same point $\vec{z}$, specifically, f_i is evaluated on $\vec{z}_{[i+1:]}$. We highlight the modifications to DeepFold.Eval in orange.

$\mathsf{BatchEval}(\mathsf{pp}, \{\mathcal{C}_i, y_i, \mu_i, \tilde{f}_i, \mathcal{D}_i\}_{i=0}^{\mu}, \vec{z}) \rightarrow b$: We assume $\tilde{f}_i$ ($i \in [0, \mu-1]$) is a $(\mu-i)$-variate multilinear polynomial. Given commitments $\mathcal{C}_i = \langle rt_i, \alpha^{2^i}, c_i \rangle$, the evaluation point $\vec{z}$, $y_i = \tilde{f}_i(\vec{z}_{[i+1:]})$, $\{\tilde{f}_i\}$, $\mathcal{P}$ and $\mathcal{V}$ run the following:

1: Let $A_0 := \{\vec{z}, \vec{\alpha}\}$, and $\vec{\alpha} = (\alpha^{2^0}, \alpha^{2^1}, \cdots, \alpha^{2^{\mu-1}})$. For $i \in [\mu-1]$, let $c_{\vec{z}_{[i+1:]}} := y_i$, $c_{\vec{\alpha}_{[i+1:]}} := c_i$.

2: For each round i, where $i \in [\mu]$,

- a. $\mathcal{V}$ sends $\alpha_i \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$. Let $A_{i-1} := \{A_{i-1}, \vec{\alpha}_i\}$, where $\vec{\alpha}_i = (\alpha_i^{2^0}, \alpha_i^{2^1}, \cdots, \alpha_i^{2^{\mu-i}})$.
- b. If $i = \mu$, $\mathcal{P}$ sends $g(X) := \tilde{f}(r_1, ..., r_{i-1}, X)$ to $\mathcal{V}$. Otherwise, let $A_i := \emptyset$, for each $\vec{w} \in A_{i-1}$, $\mathcal{P}$ sends $g_{\vec{w}_{[2:]}}(X) := \tilde{f}^{(i-1)}(X, \vec{w}_{[2:]})$ to $\mathcal{V}$. Let $A_i \leftarrow \{A_i, \vec{w}_{[2:]}\}$. For $j \in [i, \mu-1]$, $\mathcal{P}$ sends $c_{\vec{w}} := f_j(\alpha_i^{2^{j-i+1}})$ to $\mathcal{V}$, where $\vec{w} := (\alpha_i^{2^{j-i+1}}, \cdots, \alpha_i^{2^{\mu-i}})$.
- c. $\mathcal{V}$ sends $r_i, \gamma_i \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
- d. $\mathcal{P}$ computes $f^{(i)}(X) = f_E^{(i)}(X) + r_i \cdot f_O^{(i)}(X) + \gamma_i \cdot f_i(X)$, where $f_E^{(i)}(X), f_O^{(i)}(X)$ satisfy that:
$$f^{(i-1)}(X) = f_E^{(i)}(X^2) + X \cdot f_O^{(i)}(X^2)$$
- e. Let $\vec{v}^{(i)} = f^{(i)}|_{L_i}$. If $i = \mu$, $\mathcal{P}$ sends a scalar $f^{(\mu)} \in \mathbb{F}$ to $\mathcal{V}$. Otherwise, $\mathcal{P}$ sends $\mathsf{MT.Commit}(\vec{v}^{(i)}) \rightarrow rt_i$.

3: $\mathcal{V}$ checks $g_{\vec{z}_{[2:]}}(z_1) = y_1$, $g_{\vec{\alpha}_{[2:]}}(\alpha) = c_1$, $g(r_\mu) = f^{(\mu)}$. For each round i, where $i \in [\mu]$,

- a. For each $\vec{w} \in A_{i-1}$, if $i < \mu$, $\mathcal{V}$ checks $g_{\vec{w}}(r_i) + \gamma_i \cdot c_{\vec{w}_{[2:]}} = g_{\vec{w}_{[2:]}}(w_1)$; otherwise, $\mathcal{V}$ checks $g_{\vec{w}}(r_i) = g(w_1)$.

4: Repeat the following query process s times:

- a. $\mathcal{V}$ sends $\beta_0 \xleftarrow{\$} L_0$ to $\mathcal{P}$. For $i \in [\mu]$, define $\beta_i := \beta_{i-1}^2$.
- b. For $i \in [\mu]$, $\mathcal{P}$ opens $f^{(i-1)}(\pm\beta_{i-1})$ and $f_i(\beta_i)$ using $\mathsf{MT.Open}$.
- c. $\mathcal{V}$ checks the results using $\mathsf{MT.Verify}$.
- d. For $i \in [\mu]$, $\mathcal{V}$ checks the following triple are on a common line.
$$\left(\beta_i, f^{(i)}(\beta_i)\right), \left(-\beta_i, f^{(i)}(-\beta_i)\right)$$
$$\left(r_{i+1}, f^{(j+1)}(\beta_{j+1}) - \gamma_{i+1} \cdot f_{j+1}(\beta_{j+1})\right)$$

5: If all above checks pass, $\mathcal{V}$ outputs 1; 0 otherwise.

Complexity analysis. It is easy to observe that the batch evaluation protocol also has linear prover time and poly-logarithmic verifier time and proof size. It outperforms the naive approach of evaluating each polynomial separately, which results in a proof size of $\sum_{i=1}^{\mu} O(i^2) = O(\log^3 n)$.

Soundness. The soundness error of DeepFold.BatchEval is $\varepsilon = \mathsf{poly}(|L_0|)/|\mathbb{F}| + (1-\Delta)^s$, derived from the following:

Theorem 2. *If the verifier outputs* 1 *with probability greater than* $\mathsf{poly}(|L_0|)/|\mathbb{F}| + (1-\Delta)^s$, *there exist multilinear polynomials* $\{\tilde{p}_i(X_{i+1}, \cdots, X_\mu)\}_{i=0}^{\mu-1}$ *such that* $\tilde{p}_i(\vec{z}_{[i+1:\mu]}) = y_i$, $\tilde{p}_i(\vec{\alpha}_{[i+1:\mu]}) = c_i$ *and* $\mathsf{Ham}(f^{(i)}|_{L_i}, p_i|_{L_i}) < \Delta$ *for* $i \in [0, \mu-1]$. *Let* $\tilde{p}^{(0)} = \tilde{p}_0$, *and there exist polynomials* $\tilde{p}^{(i)}$ *such that*

$$\tilde{p}^{(i)}(X_{[i+1:\mu]}) = \tilde{p}^{(i-1)}(r_i, X_{[i+1:\mu]}) + \gamma_i \cdot \tilde{p}_i(X_{[i+1:\mu]}) \quad (4)$$

for $i \in [\mu]$, where $\mathsf{Ham}(f^{(i)}|_{L_i}, p^{(i)}|_{L_i}) < \Delta$ for $i \in [0,\mu]$.

We refer to the derivation of $p^{(i)}$ from $p^{(i-1)}$ and p_i in Equation 4 as the ***folding-batch*** *of $p^{(i-1)}$ and p_i.*

Proof. The proof is provided in Appendix C. The key to address the problem posed by the list decoding radius is Lemma 7, which shows that an additional evaluation can uniquely determine the polynomial within the list decoding radius. □

Batch evaluation on different points. Our technique can be extended to batch evaluation on different points, namely, $c_i = f_i(\alpha_i)$ and $y_i = \tilde{f}_i(\vec{z}_i)$, where $\{\alpha_i\}, \{\vec{z}_i\}$ are unrelated to each other. The idea is to first run a batched sumcheck [34] protocol to reduce each evaluation to $\tilde{f}_i(\vec{r}_{[i+1:]})$, where $\vec{r}$ is the shared random challenge from $\mathcal{V}$. We can now apply the above protocol directly to $\{\tilde{f}_i(\vec{r}_{[i+1:]})\}$. We provide the batch sumcheck protocol and the related ideas in Appendix D.

5.2 Achieving Zero-knowledge for DeepFold

With the batch evaluation technique at hand, we delve into the zero-knowledge property of DeepFold. Zero-knowledge requires that the PCS does not expose any information about the underlying $\tilde{f}$. The zero-knowledge property is much easier to incorporate for homomorphic commitment schemes like mKZG [35]. For example, to open $\tilde{f}(\vec{z})$, $\mathcal{P}$ can commit to some random polynomial $\tilde{g}$ and prove $(\tilde{f} + r \cdot \tilde{g})(\vec{z})$ for some random $r \in \mathbb{F}$. Unfortunately, this technique cannot be applied directly to our hash-based scheme. Virgo proposes a clever zk-PCS based on FRI, but their commitment scheme is completely different from DeepFold, making it hard to adapt.

Before presenting our construction, we first analyze what information has been leaked in DeepFold: (i) $\mathcal{V}$ can inquire about $f^{(i)}$ at some point in each round, and (ii) $\mathcal{V}$ has access to $f^{(i)}|_{L_i}$ at certain locations during the query phase. A traditional approach is to mask f with some random polynomials [45]. However, this does not help conceal certain parts of the commitment, such as the evaluation $c := \tilde{f}^{(0)}(\alpha)$ and the leaves of the Merkle trees.

To mitigate this, $\mathcal{P}$ can employ a technique by augmenting the μ-variate multilinear polynomial $\tilde{f}$ into a $\mu+1$-variate polynomial with 2^μ additional random coefficients, denoted as $\tilde{f}_{ext}$, such that $\tilde{f}_{ext}(\vec{z}||0) = \tilde{f}(\vec{z})$. Then, for each $x \in \mathbb{F}\backslash\{0\}$, $f_{ext}(x)$ would be a random value, masked by the random coefficients, where f_{ext} is the twin polynomial of $\tilde{f}_{ext}$. Nonetheless, this augmentation is still not enough. Intuitively, during the evaluation, $\mathcal{P}$ will send $\tilde{f}_{ext}(r_1, \cdots, r_\mu, X)$ in the last round. By replacing X with zero, $\mathcal{V}$ can obtain the evaluation of $\tilde{f}(r_1, \cdots, r_\mu)$, which essentially leaks some information. In fact, there is also leakage in the last few rounds of the evaluation, as $\mathcal{V}$ can open s points on $f_{ext}^{(i)}|_{L_i}$, which is sufficient for $\mathcal{V}$ to interpolate the entire polynomial. $\mathcal{V}$ can immediately infer some information about $\tilde{f}$ from the first half of $f_{ext}^{(i)}$'s coefficients, as they are not masked by randomness.

To address this, we propose that $\mathcal{P}$ use the batching technique from Section 5.1 to open $\tilde{f}_{ext}$ together with another random polynomial, denoted by $\tilde{g}$. Since the leakage occurs only in the last few rounds, $\tilde{g}$ can be much smaller than $\tilde{f}_{ext}$, hiding $\tilde{f}_{ext}$ in the last ℓ rounds. Thus, this modification does not significantly affect efficiency. We summarize our construction by making a slight modification to DeepFold and obtain zkDeepFold, which is formalized as follows:

- $\mathsf{Commit}(\mathsf{pp}, \tilde{f}) \rightarrow (\mathcal{C}, \mathcal{D})$: Let $\vec{f}$ denote the coefficients of $\tilde{f}$. $\mathcal{P}$ samples $\vec{r} \xleftarrow{\$} \mathbb{F}^{2^\mu}$. Let $\tilde{f}_{ext}$ denote a $(\mu+1)$-variable multilinear polynomial whose coefficient vector is $\vec{f}_{ext}$, where $\vec{f}_{ext} = \vec{f}||\vec{r}$. Let $\tilde{g}$ denote a random ℓ-variable multilinear polynomial, such that $2^\ell > s \cdot \mu^2$. $\mathcal{P}$ invokes $\mathsf{DeepFold.Commit}$ to commit to $\tilde{f}_{ext}$ and $\tilde{g}$.
- $\mathsf{Eval}(\mathsf{pp}, \mathcal{C}, \vec{z}, y, \mu, \tilde{f}, \mathcal{D}) \rightarrow b$: $\mathcal{P}, \mathcal{V}$ run the $\mathsf{BatchEval}$ protocol for $y = \tilde{f}_{ext}(\vec{z}||0)$ and $\tilde{g}(\vec{z}_{[\mu-\ell+2:]}||0)$.

Theorem 3. *$\mathsf{zkDeepFold}$ is a zero-knowledge polynomial commitment scheme.*

Proof. The definition of zero-knowledge and the construction of the simulator are presented in Appendix E. □

6 Applications to Zero-knowledge Proofs

In this section, we discuss several applications of our novel multilinear polynomial commitment scheme. Our techniques can be effectively combined with these practical applications, resulting in significant improvements in actual efficiency.

6.1 zk-SNARK Systems

Modern zk-SNARKs [18, 45] often adopt a recipe by combining a multilinear PCS with a PIOP, such as Libra [45] and HyperPlonk [18], to obtain a prover-efficient proof system. For example, in Libra [45], the prover $\mathcal{P}$ encodes the witness $\vec{f}$ in the input layer of a layered arithmetic circuit as a multilinear polynomial $\tilde{f}$. $\mathcal{P}$ then commits to $\tilde{f}$ using a multilinear PCS and later evaluates $\tilde{f}$ at a random point $\vec{u}$. In the HyperPlonk PIOP [18], the prover must commit to the entire witness in the arithmetic circuit and later evaluate it at different random points. Originally, the multilinear PCS was instantiated by schemes like mKZG [35] and Virgo [51], which either suffer from a trusted setup or evaluation inefficiency.

DeepFold serve as a substitution for instantiating the multilinear PCS. Due to its transparency, concise proof size, and prover efficiency, combining the PIOPs with DeepFold can immediately optimize the current proof systems of Libra and HyperPlonk. We provide an experimental benchmark of this instantiation against other PCS in Section 7.2.

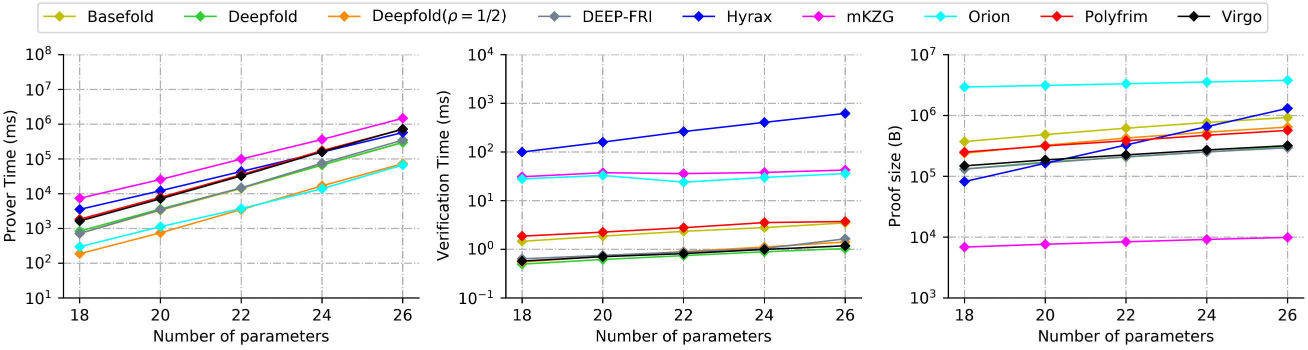

Figure 5: Performance comparison for multilinear PCS.

6.2 Verifiable Matrix Multiplications

In [43], Thaler proposes a method to prove the correctness of an $n \times n$ matrix multiplication with only $O(n^2)$ complexity, which is even lower than the computation of the multiplication itself. Let $A \in \mathbb{F}^{m\times n}$, $B \in \mathbb{F}^{n\times p}$, and $C \in \mathbb{F}^{m\times p}$. To prove that $A \times B = C$, the prover $\mathcal{P}$ first encodes the matrices into multilinear polynomials $\tilde{A}(\vec{x},\vec{y})$, $\tilde{B}(\vec{y},\vec{z})$, and $\tilde{C}(\vec{x},\vec{z})$, where $2^{|\vec{x}|} = m$, $2^{|\vec{y}|} = n$, and $2^{|\vec{z}|} = p$. Given random challenges $r_1 \in \mathbb{F}^{|\vec{x}|}$ and $r_2 \in \mathbb{F}^{|\vec{z}|}$ selected by $\mathcal{V}$, $\mathcal{P}$ only needs to invoke a sumcheck argument to prove that $\sum_{\vec{y}\in\{0,1\}^{\log n}} \tilde{A}(r_1,\vec{y})\cdot\tilde{B}(\vec{y},r_2) = \tilde{C}(r_1,r_2)$. During the argument, the prover needs to evaluate the polynomials $\tilde{A}$ and $\tilde{B}$ at some challenge points. To make the argument succinct, one can combine this argument with a multilinear PCS, which can be effectively instantiated by DeepFold.

One problem is that, in practice, the dimensions of the involved matrices are not always powers of 2. For example, in machine learning applications like GPT-2 [36], there is a need multiplying an attention matrix of size 768 × 2304. This task of arbitrary size matrix multiplication is important for zkML [42]. Naively, one could pad the matrix to the nearest size where the dimensions are all powers of 2, i.e., 1024 × 4096. However, this would result in more than 60% of the matrix being zero, leading to a significant waste when committing to and evaluating it with DeepFold. Our batching technique from Section 5 can be applied effectively: we can commit to 4 smaller matrices with dimensions 512 × 2048, 256 × 2048, 512 × 256, and 256 × 256, respectively, defined as $\tilde{A}_1, \tilde{A}_2, \tilde{A}_3$, and $\tilde{A}_4$. When evaluating $\tilde{A}(\vec{x},\vec{z})$, $\mathcal{P}$ can alternatively batch evaluate

$$y_1 = \tilde{A}_1(\vec{x}[1..9],\vec{z}[1..11]), \quad y_2 = \tilde{A}_2(\vec{x}[1..8],\vec{z}[1..11])$$
$$y_3 = \tilde{A}_3(\vec{x}[1..9],\vec{z}[1..8]), \quad y_4 = \tilde{A}_4(\vec{x}[1..8],\vec{z}[1..8])$$

and then let $\mathcal{V}$ compute $\tilde{A}(\vec{x},\vec{y})$ from y_1, y_2, y_3, and y_4. In Section 7.3, our experiments show that this technique significantly reduces both the proving cost and the proof size.

7 Experimental Evaluation

We implemented DeepFold in Rust with approximately 1,500 lines of code, and the two applications, zk-SNARKs and verifiable matrix multiplication, with 4,000 lines. Our code is published at https://doi.org/10.5281/zenodo.14725141 and https://doi.org/10.5281/zenodo.14725129.

Experiment setup. All experiments were conducted on an Intel(R) Xeon(R) Platinum 8460Y+ with a speed of 2 GHz. The server had 190GB of RAM and operated with Ubuntu 22.04 LTS. All reported figures are the averages from 10 executions. No parallelization was employed in our experiments.

7.1 Performance for DeepFold

To demonstrate the efficiency of DeepFold, we first compared the concrete performance of DeepFold against other multilinear PCS, namely mKZG [35], Hyrax [44], BaseFold [48], Virgo [51], PolyFRIM [52], and Orion [47]. We also compare the performance of DEEP-FRI univariate PCS for a more thorough analysis.

The security parameter for all schemes is set to $\lambda = 100$-bits. The number of variables μ in the multilinear polynomials ranges from 18 to 26. Implementation of RS-based schemes are all based on the framework of PolyFRIM [52] [1], which implements Virgo and PolyFRIM. We implement Basefold, DEEP-FRI and Deepfold on top of that, ensuring a fair comparison among these schemes. We use the field $\mathbb{F}_{p^2}$, where $p = 2^{61} - 1$, with a multiplicative coset of size up to 2^{60}. The hash function employed is blake3. For RS code-based schemes, we set the code rate to $\rho = \frac{1}{8}$, as in the default setting of Plonky2 [2]. We also list a more efficient version of DeepFold by setting the code rate to $\rho = \frac{1}{2}$. mKZG benchmark is from https://github.com/EspressoSystems/hyperplonk, Hyrax is from https://github.com/arkworks-rs/poly-commit, and Orion is from https://github.com/sunblaze-ucb/

[1] https://github.com/guo-yanpei/PolyFRIM

Orion. All these libraries are state-of-the-art implementations of the corresponding PCS. The results are presented in Figure 5, including prover time (for both commitment and evaluation), verifier time, and proof size.

We observe that when $\mu = 22$ and $\Delta = 1 - \rho - \varepsilon$, the prover time for DeepFold is only 16 seconds, which is 3× faster than Virgo and Hyrax, and 5× faster than mKZG. The proof size of DeepFold is 208 KB, while the proof size of BaseFold is 619 KB. This optimization is a result of the reduced query complexity in DeepFold ($s = 34$) compared to BaseFold ($s = 120$). Compared with PolyFRIM, DeepFold achieves 2× better prover time, verifier time, and proof size. This improvement is because PolyFRIM requires committing additional μ polynomials during evaluation, essentially equals running FRI twice. Compared with DEEP-FRI univariate PCS, we can observe DeepFold has almost the same prover time, verifier time and proof size.

We also observe that when setting the code rate to $\rho = \frac{1}{2}$, the prover time of DeepFold is only 3 second for a 22-variate, almost equivalent to Orion, but the proof size is 7.5× smaller. This result is a bit surprising, as DeepFold requires $O(n \log n)$ time complexity for the commitment algorithm, while Orion is known for its linear prover time. However, we note that its prover time is concretely large: the asymptotic advantage of Orion emerges only when $\mu \geq 26$, with the prover running 10% faster than DeepFold. On the other hand, we note that the concrete proof size of DeepFold is more than 10× better than Orion.

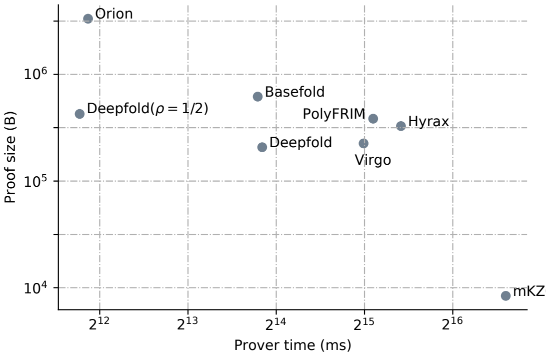

Figure 6: Trade-off of multilinear PCSs

Discussion of results. Due to the intricacy of all schemes, we present the performance of different multilinear PCSs for $\mu = 22$ in Fig. 6 to clarify their tradeoffs. We observe that DeepFold ($\rho = 1/2$) has one of the *fastest* prover times among all the benched schemes, slightly slower than BaseFold (which can also be set to $\rho = 1/2$), and is concretely competitive with Orion. It also has one of the smallest proof sizes among all code-based schemes. Compared to elliptic curve-based schemes, DeepFold has a smaller proof size than Hyrax due to better asymptotic complexity. mKZG has a much smaller proof size than DeepFold, at the cost of a trusted setup and the largest prover time. In practice, DeepFold can be useful in settings where prover time is more important than proof size.

7.2 Performance for zk-SNARKs

In this section, we present the concrete efficiency of using DeepFold to construct zk-SNARKs. We replace the PCS in open-source implementation [1] of Virgo [51] to benchmark Libra + DeepFold. As the open-source code of HyperPlonk is hard to adapt, we implement HyperPlonk PIOP from scratch and compile it with DeepFold to compare against mKZG [35] and BaseFold [48].

We evaluate zk-SNARKs using a Merkle tree circuit with the following setup: $\mathcal{P}$ convinces $\mathcal{V}$ that it knows the leaves of a Merkle tree that are consistent with a public Merkle root, where the hash function applied is SHA-256. The circuit consists of $2M - 1$ SHA-256 hashes for a Merkle tree with M leaves. In our report, the number of leaves ranges from 2 to 256. The circuit size of each SHA-256 circuit is roughly 2^{18} gates, and the total size of the largest Merkle tree ($M = 256$) is around 2^{26} gates. Figure 7 illustrates the performance.

When proving a Merkle tree with 256 leaves, Libra + DeepFold takes only 33 seconds, which is 2.5× faster than Virgo, while the verifier time is 10 ms, and the proof size is 235 KB, slightly better than Virgo. For HyperPlonk, it takes 826 seconds to prove a Merkle tree with 256 leaves when compiled with DeepFold, compared to 2967 seconds with mKZG, showcasing a 3.6× optimization. In the same setting, the proof size with DeepFold is about 310 KB, while BaseFold requires 1050 KB, demonstrating a 3.3× optimization. These observations confirm that our new multilinear PCS significantly enhances zk-SNARK performance compared to existing schemes.

7.3 Performance for Verifiable MatMult

We use the case of verifiable matrix multiplications to demonstrate the advantage of our batching technique. The benchmark is run on the multiplication between a matrix of size $n \times 768$ and another of size 768×2304. This is the same setting as the GPT-2 attention matrix. The size n varies from 150 to 1200, representing the length of input token vectors. We compared the following three approaches for verifiable matrix multiplication with DeepFold:

- **Naive padding**: In this approach, $\mathcal{P}$ pads the rows and columns of the matrices to the nearest power of two with zeros. For example, for a matrix of size 150×768, $\mathcal{P}$ adds 256 zero columns and 106 zero rows, transforming it into a 256×1024 matrix. Similarly, for the second matrix, $\mathcal{P}$ performs the same padding operation. Consequently, $\mathcal{P}$ needs to prove the validity of matrix multiplication for matrices

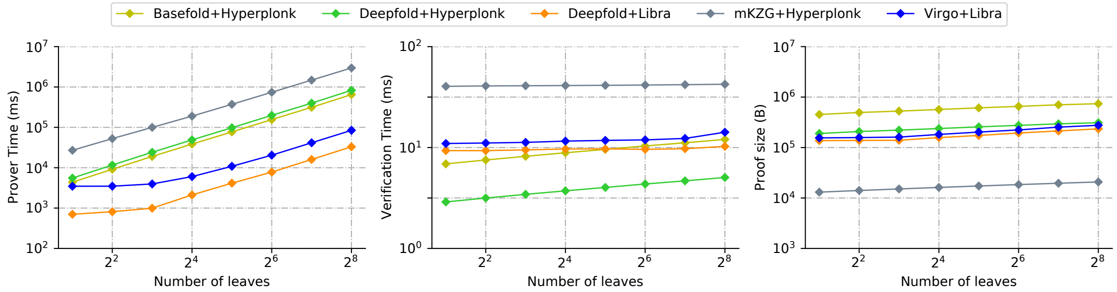

Figure 7: Performance of zk-SNARKs, when proving the knowledge of a Merkle tree

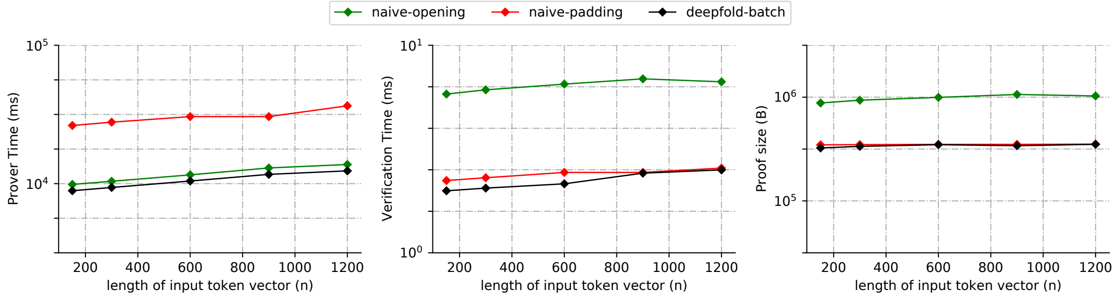

Figure 8: Performance of verifiable matrix multiplications

of sizes 256×1024 and 1024×4096, which requires multilinear PCSs with sizes $\log_2(256)+\log_2(1024) = 18$ and $\log_2(1024)+\log_2(4096) = 22$, respectively.

- **Naive opening**: $\mathcal{P}$ splits each matrix into at most four submatrices. For example, for a matrix of size 150×768, $\mathcal{P}$ splits it into 128×512, 128×256, 22×512, and 22×256, and adds 10 zero rows to the last two sub-matrices, converting them into 32×512 and 32×256, respectively. In this case, $\mathcal{P}$ requires multilinear PCSs of size 16, 15, 14 and 13. When opening, $\mathcal{P}$ needs to generate proofs for these multilinear polynomials separately.
- DeepFold **batching**: $\mathcal{P}$ splits each matrix into at most four sub-matrices as in the naive opening. The only difference is when opening those multilinear polynomials, we use batch evaluation (Section 5) on polynomials of different sizes.

Figure 8 demonstrates that the DeepFold batching technique outperforms other methods in all aspects. Specifically, the batching technique achieves approximately 2.4× better prover time compared to naive-padding, and around 3× better verifier time and proof size compared to naive-opening. This indicates that the DeepFold batching technique takes advantage of both the faster prover time of naive-opening and the smaller proof size of naive-padding.

8 Conclusion

This paper presents DeepFold, an efficient and concise multilinear polynomial commitment constructed from RS codes. DeepFold is transparent and relies solely on lightweight hash functions. The core idea is to generalize [48] to the list decoding radius setting. The main challenge is that the correlated agreement principle does not hold in this scenario. To address this problem, we incorporate an idea from [10] by adding an extra evaluation in each round and inventing a new method to validate the correctness of these evaluations.

Future works. One future direction is integrating DeepFold with industrial projects like Binius [19] and Jolt zkVM [41]. Since experiments demonstrate concrete efficiency improvements for DeepFold compared to existing multilinear PCS, we anticipate these benefits will help with real-world applications. Recently, STIR [4] proposed a method to further reduce the query repetition number in RS-IOPP. Incorporating this approach could potentially enhance the efficiency of DeepFold, making it another promising avenue for future work.

Discussion: Research Ethics

Our research focuses on enhancing the prover time and proof size of existing polynomial commitment schemes. Our method enables the prover to convincingly demonstrate the evaluation of a committed multilinear polynomial at a specific point. We strictly adhere to ethical guidelines, avoiding deceptive practices, unauthorized disclosures, live system experiments, or any actions that could compromise the well-being of our team. Consequently, our work aims to reduce the proof size of previous state-of-the-art schemes while preserving their efficient prover time.

Discussion: Open Science Policy

Our study adheres to open science principles and fully supports artifact evaluation by ensuring the availability, functionality, and reproducibility of our work. The PCS and verifiable matrix multiplication source codes are publicly accessible at https://doi.org/10.5281/zenodo.14725141, and our implementation of Hyperplonk is available at https://doi.org/10.5281/zenodo.14725129. Additionally, we have made a slight update to open source code Libra, one of our baseline, to provide more information for comparison. The updated code can be accessed at https://doi.org/10.5281/zenodo.14725139.

References

[1] https://github.com/sunblaze-ucb/Virgo.

[2] Plonky2. https://github.com/0xPolygonZero/plonky2.git.

[3] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104, 2017.

[4] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. Stir: Reed-solomon proximity testing with fewer queries. In *Annual International Cryptology Conference*, pages 380–413. Springer, 2024.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *Electron. Colloquium Comput. Complex.*, 2017.

[6] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[7] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed–solomon codes. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 900–909. IEEE, 2020.

[8] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. Aurora: Transparent succinct arguments for r1cs. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 103–128. Springer, 2019.

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

[10] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. Deep-fri: sampling outside the box improves soundness. *arXiv preprint arXiv:1903.12243*, 2019.

[11] Eli Ben-Sasson, Swastik Kopparty, and Jaikumar Radhakrishnan. Subspace polynomials and limits to list decoding of reed–solomon codes. *IEEE Transactions on Information Theory*, 56(1):113–120, 2009.

[12] Eli Ben-Sasson, Swastik Kopparty, and Shubhangi Saraf. Worst-case to average case reductions for the distance to a code. In *33rd Computational Complexity Conference (CCC 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

[13] Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michał Zając. Fiat-shamir security of fri and related snarks. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 3–40, Singapore, 2023. Springer Nature Singapore.

[14] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orru. Gemini: Elastic snarks for diverse environments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 427–457. Springer, 2022.

[15] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020.

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

[17] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 677–706. Springer, 2020.

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

[19] Benjamin E. Diamond and Jim Posen. Polylogarithmic proofs for multilinears over binary towers. Cryptology ePrint Archive, Paper 2024/504, 2024.

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

[21] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

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

[23] Ulrich Haböck, David Levit, and Shahar Papini. Circle STARKs. Cryptology ePrint Archive, Paper 2024/278, 2024.

[24] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Frida: Data availability sampling from fri. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 289–324, Cham, 2024. Springer Nature Switzerland.

[25] Ari Juels and Burton S. Kaliski. Pors: proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 584–597, New York, NY, USA, 2007. Association for Computing Machinery.

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

[27] Assimakis A Kattis, Konstantin Panarin, and Alexander Vlasov. Redshift: transparent snarks from list polynomial commitments. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1725–1737, 2022.

[28] Neal Koblitz and Alfred J Menezes. The random oracle model: a twenty-year retrospective. *Designs, Codes and Cryptography*, 77:587–610, 2015.

[29] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023.

[30] Thomas Lavaur and Jérôme Lacan. Boomy: Batch opening of multivariate polynomial commitment. *Cryptology ePrint Archive*, 2023.

[31] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021.

[32] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 39–39, 2024.

[33] Tianyi Liu, Xiang Xie, and Yupeng Zhang. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2968–2985, 2021.

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

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

[36] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[37] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. In

2023 IEEE Symposium on Security and Privacy (SP), pages 790–808. IEEE, 2023.

[38] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

[39] David Schultz, Barbara Liskov, and Moses Liskov. Mpss: mobile proactive secret sharing. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):1–32, 2010.

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

[41] Hang Su, Qi Yang, and Zhenfei Zhang. Jolt-b: recursion friendly jolt with basefold commitment. Cryptology ePrint Archive, Paper 2024/1131, 2024.

[42] Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models. *arXiv preprint arXiv:2404.16109*, 2024.

[43] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Annual Cryptology Conference*, pages 71–89. Springer, 2013.

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

[45] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 733–764. Springer, 2019.

[46] 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 *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3003–3017, 2022.

[47] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *Annual International Cryptology Conference*, pages 299–328. Springer, 2022.

[48] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: efficient field-agnostic polynomial commitment schemes from foldable codes. In *Annual International Cryptology Conference*, pages 138–169. Springer, 2024.

[49] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2039–2053, 2020.

[50] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. Polynomial commitment with a {One-to-Many} prover and applications. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2965–2982, 2022.

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

[52] Zongyang Zhang, Weihan Li, Yanpei Guo, Kexin Shi, Sherman SM Chow, Ximeng Liu, and Jin Dong. Fast {RS-IOP} multivariate polynomial commitments and verifiable secret sharing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3187–3204, 2024.

A Definition of Polynomial Commitment

The PCS satisfies *completeness* if for any multilinear polynomial $\tilde{f}_i \in \mathbb{F}[X_1, \cdots, X_{\mu_i}]$ and any point $\vec{z}_i \in \mathbb{F}^{\mu_i}$, the following probability is 1:

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, \mu) \\ (\mathcal{C}, \mathcal{D}) \leftarrow \mathsf{Commit}(\mathsf{pp}, \tilde{f}) \end{array} : \mathsf{Eval}(\mathsf{pp}, \mathcal{C}, \vec{z}, \tilde{f}(\vec{z}); \tilde{f}, \mathcal{D}) = 1\right]$$

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

$$\Pr\left[\begin{array}{r|r} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, \mu) \\ b_0 = b_1 = 1 \wedge & (\mathcal{C}, \tilde{f}_0, \mathcal{D}_0, \tilde{f}_1, \mathcal{D}_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \tilde{f}_0 \neq \tilde{f}_1 & b_0 \leftarrow \mathsf{OpenPoly}(\mathsf{pp}, \mathcal{C}, \tilde{f}_0, \mathcal{D}_0) \\ & b_1 \leftarrow \mathsf{OpenPoly}(\mathsf{pp}, \mathcal{C}, \tilde{f}_1, \mathcal{D}_1) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

The scheme satisfies *(knowledge) soundness* if Eval is an argument (of knowledge) for the relation $\mathcal{R}_{\mathsf{Eval},\mathsf{pp}}$ defined as follows:

$$\left\{(x = \{\mathcal{C}, \vec{z}, y\}); w = \{\tilde{f}, \mathcal{D}\} : \begin{array}{r} \tilde{f}(\vec{z}) = y \\ \mathsf{OpenPoly}(\mathsf{pp}, \mathcal{C}, \tilde{f}, \mathcal{D}) = 1 \end{array}\right\}$$

B Security Proofs for DeepFold

Completeness. The completeness follows directly from the construction of DeepFold.

Binding. The binding property of DeepFold comes from the following theorem.

Theorem 4. *For any committed vector $\vec{v}$ of length n and a random point $\alpha \in \mathbb{F}$, the probability that there exist two different polynomials p_1 and p_2 with bounded degree, such that $\mathsf{Ham}(p_1|_{L_0}, \vec{v}), \mathsf{Ham}(p_2|_{L_0}, \vec{v}) < 1 - \rho - \varepsilon$ and $p_1(\alpha) = p_2(\alpha)$, is negligible.*

Proof. According to Conjecture 1, define $\mathcal{L} = \mathsf{List}(\vec{v}, 1 - \rho - \varepsilon)$, where $|\mathcal{L}| \leq \mathsf{poly}(n)$. Thus, the probability that there exist such polynomials $p_1, p_2 \in \mathcal{L}$ is less than $n \cdot \binom{|\mathcal{L}|}{2} / |\mathbb{F}|$. The numerator represents the union bound of equal evaluations for all pairs of polynomials in $\mathcal{L}$. □

Soundness. The soundness is a special case of Theorem 2, which is proved in Appendix C.

C Proof of Theorem 2

First of all, we clarify some notions that have different meaning with previous sections, because of the case of malicious prover. The committed vector in the i-th round could be represented by mapping $f^{(i)} : L_i \to \mathbb{F}$. Similarly, $f_i : L_i \to \mathbb{F}$ denotes the vector representing the commitment of polynomial $\tilde{f}_i$. $f_E^{(i+1)}, f_O^{(i+1)} : L_{i+1} \to \mathbb{F}$ is defined as

$$f_E^{(i+1)}(x^2) = \frac{f^{(i)}(x) + f^{(i)}(-x)}{2}$$
$$f_O^{(i+1)}(x^2) = \frac{f^{(i)}(x) - f^{(i)}(-x)}{2x}$$

for each $x \in L_i$. To demonstrate round-by-round soundness, we need to consider malicious case such that $f^{(i)}$ isn't equal to $f_E^{(i)} + r_i f_O^{(i)} + \gamma_i f_i$.

We prove the theorem via mathematical induction. When $\mu = 1$, this theorem holds straightforwardly. If the theorem holds for $\mu = t$, we will prove it also holds for $\mu = t + 1$.

Since the protocol from the second round to the last can be viewed as a t-variate polynomial batch evaluation, so $f^{(1)}$ must approximate some $g^{(1)}|_{L_1}$ of degree 2^t, and for each $i \in [2, \mu]$, f_i must approximate some $g_i|_{L_i}$ polynomial of degree $2^{\mu - i}$ satisfying specific evaluations based on induction assumption. For each $i \in [1, \mu - 1]$, $g^{(i+1)}$ is defined as the folding-batch of $g^{(i)}$ and g_{i+1}. Then $f^{(i)}$ is close to $g^{(i)}|_{L_i}$.

Before further proof, we first define following notions.

- $RS'_i = \{p(X) \in \mathsf{RS}[L_i, \rho] \wedge \forall \vec{w} \in A_i, \tilde{p}(\vec{v})$ equals to the specific value$\}$. From induction assumption, for each $i \in [\mu]$, $g^{(i)} \in RS'_i$.
- $I_i = \{x \in L_i | f^{(i)}(x) = g^{(i)}(x)\}$ denotes the vanishing set of $f^{(i)} - g^{(i)}|_{L_i}$.
- $\{ker_i\}_{i \in [0,\mu]}$ is a family of mappings defined recursively. Let $ker_0(S) = S$ for $S \subset L_0$, and $ker_i(S) = ker_{i-1}(\{x \in L_{i-1} | x^2 \in S\})$ for $S \subset L_{i-1}$. Intuitively speaking, $ker_i(S)$ can be viewed as the "preimages" of S in L_0 under the i-iteration of square function.
- $\{err_i\}_{i \in [0,\mu]}$ is a family of sets defined recursively. Let $err_0 = \phi$ and $err_i = err_{i-1} \cup ker_i(\{x \in L_i | f_E^{(i)}(x) + r_i \cdot f_O^{(i)}(x) + \gamma_i \cdot f_i(x) \neq f^{(i)}(x)\})$. Intuitively speaking, the latter component of err_i can be viewed as the "preimages" of points which expose "evaluation errors", and therefore, err_i is the set of all points exposing "evaluation errors" until the i-th round.
- $\mathcal{E}_i : S \mapsto \frac{|\ker_i(S) \cap err_i|}{|L_0|} + \left(1 - \frac{|S|}{|L_i|}\right)$ for $S \subseteq L_i$.

Intuitively speaking, the former component of $\mathcal{E}_i(I_i)$ can be viewed as the ratio of current "erronous" points, while the latter one is the ratio of the potential "erronous" points. Therefore, this mapping denotes a total ratio of the number of "erronous" points. In the last round, all errors are exposed, so $\mathcal{E}_\mu(I_\mu)$ means the proportion in L_0 that $\mathcal{V}$ can find an error. We will next prove if induction assumption doesn't hold, it has $\mathcal{E}_\mu(I_\mu) \geq 1 - \rho - \varepsilon$ with overwhelming probability.

We first prove following useful lemmas:

Lemma 4. *For any $S_1 \subseteq S_2 \subseteq L_i$, $\mathcal{E}_i(S_1) \geq \mathcal{E}_i(S_2)$.*

Proof. By definition,

$$\begin{aligned}
&\mathcal{E}_i(S_1) - \mathcal{E}_i(S_2) \\
=&\frac{(|S_2| - |S_1|)}{|L_i|} - \frac{(|\ker_i(S_2) \cap err_i| - |\ker_i(S_1) \cap err_i|)}{|L_0|} \\
=&|S_2 \backslash S_1| / |L_i| - |(\ker_i(S_2) \backslash \ker_i(S_1)) \cap err_i| / |\ker_i(L_i)| \\
=&\frac{|\ker_i(S_2 \backslash S_1)|}{|\ker(L_i)|} - \frac{|\ker_i(S_2 \backslash S_1) \cap err_i|}{|\ker_i(L_i)|} \geq 0
\end{aligned}$$

□

Lemma 5. *For every $i \in [\mu]$, $\theta \in [0, 1]$, if $\mathsf{Ham}(f_E^{(i)} + r_i \cdot f_O^{(i)} + \gamma_i \cdot f_i, g^{(i)}|_{L_i}) \geq \theta$, then for each $f^{(i)}$, there has $\mathcal{E}_i(I_i) \geq \theta$.*

Proof. Let $f \leftarrow f_E^{(i)} + r_i \cdot f_O^{(i)} + \gamma_i \cdot f_i$. Define set $B = I_i \cup \{x \in L_i | f(x) \neq f^{(i)}(x)\}$. Define polynomial $f' : L_i \to \mathbb{F}$ such that $f'(x) = g^{(i)}(x)$ when $x \in B$ and $f'(x) = f(x)$ when $x \notin B$.

For all $x \in B$, if $f(x) \neq f'(x)$, there has $f(x) \neq g^{(i)}(x)$. Besides, for all $x \notin B$, $f(x) = g^{(i)}(x)$. Thus, the number of inconsistency points in B is not less than the Hamming distance between $f|_{L_i}$ and $f'|_{L_i}$, i.e., $|\ker_i(B) \cap err_i| / |L_0| \geq \Delta(f|_{L_i}, f'|_{L_i})$.

Also, as $f'(x) = g^{(i)}(x)$ holds for all $x \in B$, there has $1 - |B|/|L_i| \geq \mathsf{Ham}(f'|_{L_i}, g^{(i)}|_{L_i})$. Thus, $\mathcal{E}_i(B) \geq \mathsf{Ham}(f, f') + \mathsf{Ham}(f', g^{(i)}|_{L_i}) \geq \mathsf{Ham}(f, g^{(i)}|_{L_i}) \geq \theta$. As $I_i \subseteq B$, $\mathcal{E}_i(I_i) \geq \mathcal{E}_i(B) \geq \theta$. □

Next, we will prove the following two statements: 1). The induction assumption holds for $\mu = t + 1$ or $\mathcal{E}_1(I_1) \geq 1 - \varepsilon - \rho$ and 2). For each $i \in [\mu - 1]$, with overwhelming probability $\mathcal{E}_{i+1}(I_{i+1}) \geq \mathcal{E}_i(I_i)$.

Lemma 6. *If there exist no polynomials $g^{(0)}$, g_1 satisfying the claimed evaluations and $\mathsf{Ham}(g^{(0)}|_{L_0}, f^{(0)}), \mathsf{Ham}(g_1|_{L_1}, f_1) < 1-\rho-\varepsilon$ and $g^{(1)}$ equals the folding batch of $g^{(0)}$ and g_1, there has $\mathcal{E}_1(I_1) \geq 1-\rho-\varepsilon$ with overwhelming probability.*

Now we discuss the two cases in the first round.

Case 1: For random r, α, if $f_E^{(1)} + r_1 \cdot f_O^{(1)} + \gamma_1 \cdot f_1$ approximates some polynomial in RS_1' with negligible probability, there has $\mathcal{E}_1(I_1) \geq 1-\rho-\varepsilon$ with overwhelming probability from lemma 5, given that $g^{(1)} \in RS_1'$.

Case 2: If $f_E^{(1)} + r_1 \cdot f_O^{(1)} + \gamma_1 \cdot f_1$ approximates some polynomial in RS_1' with non-negligible probability, from the DEEP theorem, $f_E^{(1)}, f_O^{(1)}, f_1$ should approximate $p_E^{(1)}, p_O^{(1)}, p_1$ satisfying claimed evaluations on α_1^2 and $\vec{z}_{[2:]}$ respectively. The following lemma shows that $f_E^{(1)} + r_1 \cdot f_O^{(1)} + \gamma_1 \cdot f_1$ cannot approximate any polynomial in RS_1' other than $p_E^{(1)} + r_1 \cdot p_O^{(1)} + \gamma_1 \cdot p_1$ with non-negligible probability. Thus, if $g^{(1)} \neq p_E^{(1)} + r_1 \cdot p_O^{(1)} + \gamma_1 \cdot p_1$, there has $E_1(I_1) \geq 1-\rho-\varepsilon$.

Lemma 7. *Let $f = f_E^{(1)} + r_1 \cdot f_O^{(1)} + \gamma_1 \cdot f_1$. For a random $\alpha \in \mathbb{F}$, if there are $p, p' \in \mathsf{RS}[L_1, \rho]$ such that $\max(\mathsf{Ham}(p, f), \mathsf{Ham}(p', f)) < 1-\rho-\varepsilon$, and that both $p(\alpha^2), p'(\alpha^2)$ equal to the combination of claimed $f(\alpha^2)$, there holds $p = p'$.*

Proof. Since $\max(\mathsf{Ham}(p, f), \mathsf{Ham}(p', f)) < 1-\rho-\varepsilon$, we have $p, p' \in \mathsf{List}(f, 1-\rho-\varepsilon)$, and $p(\alpha_1^2) = p'(\alpha_1^2)$. Because of the randomness of α_1, according to Theorem 4, the probability that $p \neq p'$ is negligible. □

The statement 1 follows from the analysis of the two cases. We now prove statement 2.

Define $g_E^{(i)}, g_O^{(i)}$ such that $g^{(i-1)}(X)$ can be decomposed into $g_E^{(i)}(X^2) + X \cdot g_O^{(i)}(X^2)$. There has $g^{(i)} = g_E^{(i)} + r_i \cdot g_O^{(i)} + \gamma_i \cdot g_i$. Define $B' = \{x \in L_{i+1} | f_E^{(i+1)}(x) = g_E^{(i+1)}(x) \wedge f_O^{(i+1)}(x) = g_O^{(i+1)}(x) \wedge f_{i+1}(x) = g_{i+1}(x)\}$. By properties of random linear combination, with probability at least $1 - |L_{i+1}|/|\mathbb{F}|$, the following equality holds

$$\mathsf{Ham}\left(f_E^{(i+1)} + r_{i+1} f_O^{(i+1)} + \gamma_{i+1} f_{i+1}, g^{(i+1)}\right) = 1 - \frac{|B'|}{|L_{i+1}|}$$

Let $B = \{x \in L_i : x^2 \in B'\}$. By definition, $B \subseteq I_i$. Define set $C = I_{i+1} \cup B'$. To prove that $\mathcal{E}_{i+1}(I_{i+1}) \geq \mathcal{E}_i(I_i)$, it suffices to prove $\mathcal{E}_{i+1}(C) \geq \mathcal{E}_i(B)$ according to Lemma 4 as $B \subseteq I_i$ and $I_{i+1} \subseteq C$.

By the definition of B', for every $x \in I_{i+1} \backslash B'$,

$$f^{(i+1)}(x) = g^{(i+1)}(x) \neq f_E^{(i+1)}(x) + r_{i+1} \cdot f_O^{(i+1)}(x) + \gamma_{i+1} \cdot f_{i+1}(x)$$

Therefore, $\ker_{i+1}(I_{i+1} \backslash B') = \ker_{i+1}(C \backslash B') \subseteq err_{i+1}$. Define sets $M = \ker_i(B) \cap err_i$ and $N = \ker_{i+1}(C \backslash B')$. As $err_i \subseteq err_{i+1}$, we have

$$M \cup N \subseteq err_i \cup \ker_{i+1}(C \backslash B') \subseteq err_{i+1}. \tag{5}$$

Besides, as $B' \subseteq C$ and $\ker_i(B) = \ker_{i+1}(B')$,

$$M \cup N = (\ker_{i+1}(B') \cap err_i) \cup N \subseteq \ker_{i+1}(C). \tag{6}$$

By Equations (5) and (6),

$$M \cup N \subseteq \ker_{i+1}(C) \cap err_{i+1}. \tag{7}$$

Since $\ker_i(B) = \ker_{i+1}(B')$, there has

$$M \cap N \subseteq \ker_{i+1}(B') \cap \ker_{i+1}(C \backslash B') = \emptyset. \tag{8}$$

According to Equation (8), Equation (7) can be rewritten as

$$|\ker_{i+1}(C) \cap err_{i+1}| \geq |\ker_{i+1}(C \backslash B')| + |\ker_i(B) \cap err_i|. \tag{9}$$

Substituting

$$\frac{|\ker_{i+1}(C \backslash B')|}{|L_0|} = \frac{|C \backslash B'|}{|L_{i+1}|} = \frac{|C| - |B'|}{|L_{i+1}|} = \frac{|C|}{|L_{i+1}|} - \frac{|B|}{|L_i|}$$

in Equation (9), the following relation holds

$$\frac{|\ker_{i+1}(C) \cap err_{i+1}|}{|L_0|} - \frac{|C|}{|L_{i+1}|} \geq \frac{|\ker_i(B) \cap err_i|}{|L_0|} - \frac{|B|}{|L_i|}.$$

Therefore, $\varepsilon_{i+1}(C) \geq \varepsilon_i(B)$.

D Batch Sumcheck for Different Variable Polynomials

We present the batch sum-check algorithm for different variable polynomials, which would be a subprotocol of batch opening different points on multiple polynomials.

For simplicity, we consider the following condition without loss of generalization. Let $\{\hat{f}_i\}_{i=0}^{\mu-1}$ be a series of multivariate polynomial, where $\hat{f}_i(X_{i+1}, \cdots, X_\mu)$ is $(\mu - i)$-variable. $\mathcal{P}$ is supposed to prove for each $i \in [0, \mu-1]$, $\sum_{\vec{b} \in \{0,1\}^{\mu-i}} \hat{f}_i(\vec{b}) = y_i$.

After receiving $\{y_i\}$, $\mathcal{V}$ sends a random challenge $r \in \mathbb{F}$. The prover can run multivariate sumcheck on the following polynomial to prove all these sums.

$$\sum_{\vec{b} \in \{0,1\}^\mu} \left(\sum_{i=0}^{\mu-1} r^i \cdot \hat{f}_i(\vec{b}_{[i+1:]}) \right) = \sum_{i=0}^{\mu-1} (2r)^i y_i$$

In our case, each $\hat{f}_i$ is product of two multilinear polynomials, denoted as $\tilde{f}_i$ and $\tilde{g}_i$. In the j-th round, the prover needs to send $a_{j,t}$ for $t = \{0,1,2\}$, where

$$\begin{aligned} a_{j,t} = & \sum_{\vec{b}\in\{0,1\}^{\mu-j}} \left(\sum_{i=0}^{j-1} r^i \left(\tilde{f}_i \cdot \tilde{g}_i\right) \left(r_{[i+1,j-1]}, t, \vec{b}\right) \right. \\ & \left. + \sum_{i=j}^{\mu-1} r^i \cdot \left(\tilde{f}_i \cdot \tilde{g}_i\right) \left(\vec{b}_{[i-j:]}\right) \right) \\ = & \sum_{\vec{b}\in\{0,1\}^{\mu-j}} \sum_{i=0}^{j-1} r^i \left(\tilde{f}_i \cdot \tilde{g}_i\right) \left(r_{[i+1,j-1]}, t, \vec{b}\right) \\ & + \sum_{i=j}^{\mu-1} r^i \cdot 2^{i-j} \cdot y_i \end{aligned}$$

which can be computed in $O(k_j \cdot 2^{\mu-j})$, where k_j is the total number of polynomials with variable number equal or larger than $\mu - j + 1$. Thus, the total prover complexity is $\sum_{j=1}^{\mu} O(k_j \cdot 2^{\mu-j}) = O(\sum_{j=1}^{\mu} k_j \cdot 2^{\mu-j+1} - \sum_{j=1}^{\mu} k_{j-1} \cdot 2^{\mu-j+1}) = \sum_{j=1}^{\mu} O((k_j - k_{j-1}) \cdot 2^{\mu-j+1})$, equivalent to the total length of all polynomials.

Invoking batch sumcheck for following polynomials can reduce the evaluations of different polynomials $\tilde{f}_i$ on different points, together with different α_i in commitment to the evaluations of them on a single random point $\vec{r}$

$$\sum_{\vec{b}\in\{0,1\}^{\mu-i}} \tilde{f}_i(\vec{b}) \cdot \tilde{eq}(\vec{b}, \vec{z}^{(i)}) = y_i$$

$$\sum_{\vec{b}\in\{0,1\}^{\mu-i}} \tilde{f}_i(\vec{b}) \cdot \tilde{eq}\left(\vec{b}, (\alpha_i, \alpha_i^2, \cdots, \alpha_i^{2^{\mu-i-1}})\right) = c_i$$

E Proof of Zero-knowledge

A PC is considered to be **zero-knowledge** if there is a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all multilinear polynomial $\tilde{f}$, PPT non-uniform adversary $\mathcal{A}$:

$$\Pr\left[b = 1 \;\middle|\; \begin{array}{r} \mathcal{C} \leftarrow Commit(pp, \tilde{f}) \\ \vec{z} \leftarrow \mathcal{A}(\mathcal{C}, pp) \\ (y, tr) \leftarrow \langle \mathcal{P}(\tilde{f}), \mathcal{V} \rangle(\vec{z}) \\ b \leftarrow \mathcal{A}(\mathcal{C}, y, \vec{z}, tr) \end{array} \right] =$$

$$\Pr\left[b = 1 \;\middle|\; \begin{array}{r} \mathcal{C} \leftarrow \mathcal{S}_1(pp) \\ \vec{z} \leftarrow \mathcal{A}(\mathcal{C}, pp) \\ (y, tr) \leftarrow \mathcal{S}_2(\tilde{f}(\vec{z}), \vec{z}) \\ b \leftarrow \mathcal{A}(\mathcal{C}, y, \vec{z}, tr) \end{array} \right]$$

A simulator $\mathcal{S}$ can be constructed as following:

1. Generate a random $(\mu+1)$-variate multilinear polynomial $\tilde{f}_r$, a random $(\mu - \ell + 2)$-variate multilinear polynomial $\tilde{g}$, and use them to simulate zkDeepFold.Commit to generate the commitment $\mathcal{C} = \langle rt_0, \alpha_0, c_0, rt_1, \alpha_1, c_1 \rangle$.
2. On receiving query point $\vec{z}$ and let $y = \tilde{f}(\vec{z})$, $\mathcal{S}$ then proceeds a sum-check protocol on $\sum_{\vec{b}\in H_{\mu+1}} \hat{h}(\vec{b}) = y + r \cdot c_0 + r^2 \cdot c_1^2$, where $\hat{h}(\vec{x})$ is defined as

   $$\tilde{f}_r(\vec{x})(\tilde{eq}(\vec{z}_{ext}, \vec{x}) + r \cdot \tilde{eq}(\vec{\alpha}_0, \vec{x})) + r^2 \tilde{g}(\vec{x}_{[\ell:]}) \cdot \tilde{eq}(\vec{\alpha}_1, \vec{x}_{[\ell:]})$$

   with $\vec{z}_{ext} = \vec{z}||0$, $\vec{\alpha}_0 = (\alpha_0^{2^0}, \alpha_0^{2^1}, \cdots, \alpha_0^{2^\mu})$, $\vec{\alpha}_1 = (\alpha_1, \alpha_1^2, \cdots, \alpha_1^{2^{\mu-\ell+1}})$. Assume the random number used in the sumcheck is $\vec{z}'$, then the final claim is the value of $\hat{h}(\vec{z}')$.
3. $\mathcal{S}$ sends a set of randoms $y_1, y_2 \in \mathbb{F}$ representing $\tilde{f}_r(\vec{z}')$, $\tilde{g}(\vec{z}'_{[\ell:]})$, which satisfy the sum-check condition.
4. Given s queries, $\mathcal{S}$ interpolates multilinear polynomials $f^{(0)}, g^{(\ell-1)}$ such that $\tilde{f}^{(0)}(\vec{z}') = y_1$, $\tilde{g}^{(\ell-1)}(\vec{z}'_{[\ell:]}) = y_2$, and that $f^{(0)}$, $g^{(\ell-1)}$ match f_r, g on all s queries.
5. $\mathcal{S}$ uses $f^{(0)}$ and $g^{(\ell-1)}$ to finish the rest of batch evaluation.

Next, we will show the real transcript and the simulator transcript are indistinguishable. In step 2, $\mathcal{V}$ receives as the real transcript in the i-th ($i \le \mu$) round

$$\sum_{b_{i+1},\cdots,b_{\mu+1}\in\{0,1\}} \hat{h}(z'_1, \cdots, z'_{i-1}, X, b_{i+1}, \cdots, b_{\mu+1})$$

When $b_{\mu+1} = 1$, the evaluation of $\hat{h}$ would be uniformly random since the evaluation of $\tilde{f}_{ext}$ is random, so the real transcript is uniformly random. When $i = \mu + 1$, $\mathcal{V}$ receives $\hat{h}(z'_1, \cdots, z'_\mu, X)$, which is also uniformly random because of $\tilde{g}$'s randomness.

The evaluation of $\tilde{f}_{ext}(\vec{z}')$, i.e. y_1 sent at step 4 should be a random number because when $z'_{\mu+1} \neq 0$ (the probability is negligible) the evaluation of $\tilde{f}_{ext}(\vec{z}')$ is random. When it comes to step 5, all the scalars received by $\mathcal{V}$ in first ℓ round are evaluations on $\tilde{f}_{ext}$, and $\mathcal{V}$ cannot get any message of $\tilde{f}$ from them. After ℓ rounds, since $f^{(i)}$ is masked by $\tilde{g}$, $\mathcal{V}$ cannot distinguish it with the transcript of $\mathcal{S}$.