Efficient Sum-Check for High-Degree Polynomials

Yu Wang[1,2,3], Yuncong Zhang[1,2,3], and Yu Chen[1,2,3,4](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
[2] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao 266237, China
[3] Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Qingdao 266237, China
[4] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
ywang21@mail.sdu.edu.cn, {yuncong,yuchen}@sdu.edu.cn

Abstract

This paper presents a new framework for constructing efficient sum-check protocols for high-degree virtual multivariate polynomials. Existing approaches, such as HyperPlonk, suffer from an inherent bottleneck: polynomial multiplications inside the sum-check protocol lead to quasi-linear prover overhead in the polynomial degree d. To address this issue, we introduce doubly efficient sum-check (DESC) PIOP, which achieves linear prover time by integrating a generalized GKR protocol for general arithmetic circuits. The key insight is to reformulate virtual polynomial evaluation as a data-parallel circuit execution, thereby avoiding explicit polynomial multiplications.

Building on the DESC framework, we derive two concrete applications. First, for customizable constraint systems (CCS), we reduce prover time complexity from $O(qmd \log^2 d)$ to $O(qmd)$, where q denotes the number of constraint multisets, m the number of constraints, and d the maximum degree of any variable across all polynomials. Second, we propose SPARK PIOP, an improved polynomial IOP for sparse multilinear commitments, which reduces the total commitment size by nearly a quarter compared to the Lasso construction.

Finally, as a conceptual consolidation, we emphasize that the efficiency of sum-check–based proof systems is fundamentally determined by how algebraic constraints are organized. By reorganizing circuit layers into low-degree polynomial relations, we reduce communication complexity while preserving the doubly efficient properties of GKR for both the prover and the verifier.

Keywords: Sum-Check, High-Degree Polynomial, GKR Protocol.

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs), exemplified by schemes such as Groth16 [17], Bulletproofs [6], and Stark [2],

represent a remarkable cryptographic primitive for proving memberships in NP languages. These sophisticated cryptographic constructions enable a prover to assure a verifier of the validity of a given statement by presenting concise proofs while disclosing no additional information. With prominent applications in Zcash [21], zkRollups [7], zkBridge [30], and zkLogin [1], zkSNARKs have become crucial for enabling privacy-preserving and scalable applications, driving continued research towards improving their efficiency and applicability.

Despite their powerful asymptotic guarantees, modern zkSNARKs encounter a significant practical challenge: the computational burden on the prover. In real-world scenarios, this burden increases considerably with the complexity of the underlying computation. This growth is primarily governed by the number of constraints and the degree of the polynomials involved. As a result, enhancing the prover's efficiency remains a crucial objective for enabling wider adoption.

A recurring component in many IOP-based SNARKs is the sum-check protocol [19], which verifies that a multivariate polynomial sums to a claimed value over the Boolean hypercube. While sum-check enjoys low communication and verifier complexity, its prover complexity crucially depends on how polynomial evaluations are realized. In particular, when verifying identities involving high-degree polynomials, existing constructions such as HyperPlonk [9] require explicit computation of univariate polynomials during the sum-check protocol. This step relies on fast polynomial multiplication and introduces a quasi-linear $\log^2 d$ overhead in the polynomial degree d. Similar overheads arise in SuperSpartan [24] when extending polynomial IOPs from R1CS to Customizable Constraint Systems (CCS).

Although many SNARK constructions operate with low-degree polynomial constraints, higher-degree polynomials arise in more expressive settings. In particular, in HyperPlonk and CCS, higher-degree constraints result from the use of customized gates with elevated algebraic degree, whereas in lookup arguments [18,25], the use of rational functions (e.g., logarithmic derivatives) introduces polynomial identities of increased degree. In such settings, the effective polynomial degree may scale with the algebraic degree of customized gates or the arity of the constraints, making the $\log^2 d$ factor in existing sum-check protocols non-negligible in the prover complexity.

This work shows that the quasi-linear prover overhead in existing high-degree sum-check protocols is not inherent. At a high level, we demonstrate that this overhead can be eliminated by avoiding explicit polynomial multiplication, resulting in provers whose running time is linear in the effective degree of the polynomial.

1.1 The Contributions

We summarize our main contributions as follows.

A GKR-Based Sum-Check for High-Degree Polynomials. We introduce DESC PIOP, a new protocol for verifying sum-checks of high-degree virtual polynomials [9] over the Boolean hypercube, achieving linear prover time. Our key observation is that sum-check efficiency is not intrinsic to polynomial degree,

but rather to how algebraic constraints are organized and exposed to the GKR framework. By encoding high-degree compositions as structured arithmetic circuits and applying a generalized GKR protocol [31] over aggregated constraints, DESC PIOP bypasses the polynomial multiplication bottleneck that arises in conventional sum-check, while preserving soundness and achieving linear prover time.

CCS with Linear Prover Time via Algebraic Constraint Re-encoding. We integrate DESC PIOP into the CCS, yielding a proof system with $O(qmd)$ prover complexity, where q denotes the number of constraint multisets, m the constraint system size, and d the maximum multiset cardinality. This improves upon the original CCS implementation, which required $O(qmd \log^2 d)$ prover time. The improvement follows from a re-encoding of CCS constraints as structured compositions of sparse matrix–vector products and Hadamard products, enabling their verification via linear-time GKR-style sum-checks rather than explicit high-degree polynomial expansions.

Optimized Sparse Polynomial Commitments. We propose SPARK PIOP, a polynomial IOP for sparse multilinear polynomial commitments that improves prover performance. By combining logarithmic-derivative-based lookup arguments with our DESC PIOP framework, we reduce the total commitment size from $(3c+1) \cdot \mathsf{pc}(n) + c \cdot \mathsf{pc}(m)$ in Lasso to $(2c+2) \cdot \mathsf{pc}(n) + c \cdot \mathsf{pc}(m)$, while preserving linear prover time. Here, n denotes the number of nonzero entries, m is the table size, c is the decomposition parameter, and $\mathsf{pc}(\cdot)$ is the polynomial commitment size of a multilinear polynomial. Moreover, SPARK PIOP supports efficient batch verification of multiple lookup checks and sparse matrix evaluations in parallel, without compromising soundness. For example, within the CCS framework, this enables the prover to generate proofs for multiple sparse polynomial evaluations simultaneously, aggregating them into a single sum-check instance and thereby reducing the total commitment size.

A Refined Perspective on GKR-Style Proofs. Finally, as a conceptual consolidation, we generalize the core insight behind DESC—that constraint organization dictates efficiency—to the GKR protocol itself. While DESC (Section 3) reorganizes constraints to solve the computational bottleneck of high-degree polynomials, Section 6 extends this philosophy to address the communication bottleneck in general circuits.

By reorganizing circuit layers into low-degree polynomial relations, we show that the constraint reorganization is a universal tool for optimizing GKR-based proof systems. Specifically, for a circuit of depth d and width S, this approach reduces the total communication complexity from the standard $6d \log S$ to approximately $4.25d \log S$ ($\approx 29\%$ reduction), while preserving linear prover time and efficient verification.

1.2 Technical Overview

At a conceptual level, our work highlights a unifying principle: the efficiency of sum-check–based proof systems is determined by how algebraic constraints are organized before verification. Enforcing high-degree polynomial sum-checks

directly leads to unnecessary prover overhead. By reorganizing these constraints into low-degree polynomial forms and applying a GKR-style proof framework, we preserve linear-time prover complexity and maintain efficient verification.

Based on this principle, we summarize our technical contributions in the following four points.

Sum-Check for High-Degree Polynomials. We first apply this framework to the problem of verifying sums of high-degree multivariate polynomials over the Boolean hypercube. Specifically, we consider polynomials of the form $f(x) = h(g_1(x), ..., g_c(x))$, where h is a degree-d polynomial represented by a general arithmetic circuit and each g_i is multilinear. A naive application of sum-check would require explicitly expanding h, which would lead to prohibitive prover complexity. We introduce DESC PIOP to avoid this bottleneck. Crucially, DESC reformulates the high-degree polynomial evaluation as a data-parallel circuit: it treats h as a low-depth arithmetic subcircuit and evaluates 2^μ independent, parallel copies of it over the Boolean hypercube. The sum-check protocol is first used to reduce the global sum $\sum_{x\in\{0,1\}^\mu} f(x)$ to a single evaluation of the multilinear extension $\tilde{f}$ at a random point. The resulting evaluation is then verified using the generalized GKR protocol, without performing explicit polynomial multiplications.

Optimizing Prover Time in CCS. We next apply DESC PIOP to the CCS framework. A CCS instance enforces polynomial constraints of the form

$$\sum_{i=0}^{q-1} c_i \cdot \prod_{j\in S_i} \widetilde{g}_j(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \{0,1\}^n,$$

where each g_j is a sparse linear combination of the witness, and $\widetilde{g}_j$ denotes its multilinear extension over the Boolean hypercube. The degree of the resulting polynomial is determined by the maximum cardinality of the multisets S_i, which can be large in practice.

By interpreting the CCS constraint as a high-degree polynomial identity over the Boolean hypercube, its verification can be reduced to a single sum-check instance. DESC PIOP executes this sum-check with linear-time prover complexity by evaluating the underlying computation as a data-parallel arithmetic circuit. Consequently, the prover time of CCS is reduced from $O(qmd\log^2 d)$ to $O(qmd)$, while preserving soundness and efficient verification.

SPARK PIOP and Batch Verification for Sparse Commitments. We further apply our techniques to sparse multilinear polynomial commitments. In the SPARK protocol [23], verifying a sparse polynomial evaluation is reduced to checking lookup relations. While originally implemented via product checks, we point out that these relations can also be naturally enforced via logarithmic-derivative arguments. The advantage of this approach is that it enables efficient batching: after clearing denominators, these checks give rise to high-degree polynomial identities. By integrating logarithmic-derivative arguments with DESC PIOP, our framework inherently optimizes the batching process, significantly reducing commitment overhead without compromising soundness.

Revisiting GKR through Constraint Organization. Rather than treating sum-check as an intrinsic component of GKR, we observe that the efficiency of GKR-style proofs is governed by how circuit constraints are algebraically organized before invoking sum-check. By encoding circuit layers using low-degree polynomial constraints expressed via summations, sparse matrix–vector multiplications, and Hadamard products, additive relations are absorbed and multiplicative dependencies are aggregated.

This reorganization reduces the effective circuit depth and the number of sum-check layers, while maintaining the doubly efficient nature of GKR for both the prover and the verifier. Furthermore, by analyzing the impact of bounded polynomial degrees on communication cost, we identify degree parameters that achieve an optimal communication trade-off within our protocol.

1.3 Related Work

Modern SNARKs generally compile information-theoretic proofs (e.g., Interactive Oracle Proofs (IOPs) [4]) into cryptographic proofs via polynomial commitments. While systems like STARK [2], Aurora [3], Sonic [20], Plonk [14], and Fractal [10] rely on univariate polynomial commitment schemes, a growing number of frameworks—including Libra [29], Spartan [23], Gemini [5], HyperPlonk [9], and Brakedown [16]—leverage multilinear polynomial commitment schemes. At the foundational core of multilinear SNARKs is the sum-check [19]. Despite its powerful capabilities, applying traditional sum-check (e.g., HyperPlonk [9]) to high-degree multivariate polynomials introduces a computational bottleneck: evaluating expressions of the form $v = \sum_{\boldsymbol{x}\in\{0,1\}^\mu} \prod_{i=1}^{d} g_i(\boldsymbol{x})$ requires $O(2^\mu \cdot d \log^2 d)$ prover time and $O(d\mu)$ communication/verification. Our DESC framework (Section 3) resolves this inefficiency. By integrating the GKR protocol [15], DESC strictly improves the prover time to $O(2^\mu \cdot d)$ while maintaining highly efficient $O(\mu \log d + \log^2 d)$ communication and $O(\mu \log d + \log^2 d + d)$ verification.

While the proof systems like GKR, Plonk, and Spartan provide efficient verification for arithmetic circuits, they incur massive overheads when expressing non-arithmetic operations (e.g., range checks). Lookup arguments [13] circumvent this by enforcing relations against precomputed tables, with recent advancements achieving costs independent of table size [8,12]. Fundamentally, lookups serve as a powerful tool to enforce memory consistency. Spartan [23] introduced Memory-in-the-Head using 2-dimensional tensors, which Lasso subsequently generalized to c-dimensional tensors. While both rely on product checks, we observe the additive structure of logarithmic derivatives [18] excels in batch verification. Integrating this into DESC improves proving efficiency and reduces Lasso's commitment overhead from $4c + 1$ to $3c + 2$ polynomials (a $\sim 25\%$ reduction).

The foundational GKR protocol [15] and its extensions [29] established succinct interactive proofs with linear prover time. However, their standard evaluations are typically constrained to basic addition and multiplication operations, which significantly limits the optimization space for layer transitions. To overcome this, we optimize the GKR architecture via CCS [24] by employing custom

gates. This approach explicitly balances the number of sum-checks against the per-round polynomial degrees, thereby providing a highly flexible trade-off between prover time and communication complexity while strictly maintaining the asymptotic linear prover overhead.

2 Preliminaries

2.1 Notation

Let $\mathbb{F}$ be a finite field. We define the Boolean hypercube of dimension μ as $B_\mu := \{0,1\}^\mu \subseteq \mathbb{F}^\mu$. For any positive integer n, we denote its binary representation by $\langle n \rangle = (a_1, \ldots, a_k)$, where $k = \lceil \log_2 n \rceil$. This representation satisfies $n = \sum_{i=1}^{k} a_i \cdot 2^{i-1}$, with coefficients $a_i \in \{0,1\}$ and $\lceil \cdot \rceil$ denoting the ceiling function. Furthermore, we let $\mathcal{F}_\mu^{\leq d}$ denote the family of μ-variate polynomials in $\mathbb{F}[X_1, \ldots, X_\mu]$ such that the individual degree of each variable is at most d. Any polynomial $f \in \mathcal{F}_\mu^{\leq d}$ can be accessed via a virtual oracle, denoted as $[[f]]$.

2.2 The Multivariate Polynomial

A multivariate polynomial involves more than one variable; otherwise, it's termed a univariate polynomial. If the degree of the multivariate polynomial in each variable is at most one, it's referred to as a multilinear polynomial.

Definition 1 (Low-degree polynomial). *A multivariate polynomial f over a finite field $\mathbb{F}$ is called low-degree polynomial if the degree of f in each variable is exponentially smaller than $|\mathbb{F}|$.*

Lemma 1 (Schwartz-Zippel Lemma [22,26]). *Let $\mathbb{F}$ be any field, and let $g : \mathbb{F}^\mu \to \mathbb{F}$ be a nonzero μ-variate polynomial of total degree at most d. Then on any finite set $S \subseteq \mathbb{F}$,*

$$\Pr_{x \leftarrow S^\mu}[g(\boldsymbol{x}) = 0] \leq d/|S|.$$

Definition 2 (Low-degree Extensions (LDEs)). *Suppose $g : \{0,1\}^\mu \to \mathbb{F}$ is a function that maps μ bit elements into an element of $\mathbb{F}$. A polynomial extension of g is a low-degree μ-variate polynomial $\widetilde{g}(\cdot)$ such that $\widetilde{g}(\boldsymbol{x}) = g(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^\mu$.*

A multilinear polynomial extension (MLE), denoted as $\widetilde{g}(\cdot)$, is a low-degree polynomial where the degree of each variable is at most one. For a function $f : \{0,1\}^\mu \to \mathbb{F}$, its multilinear extension $\widetilde{f} : \mathbb{F}^\mu \to \mathbb{F}$ is the unique multilinear polynomial that extends f as follows

$$\begin{aligned}\widetilde{f}(x_1, \ldots, x_\mu) &= \sum_{e \in \{0,1\}^\mu} f(\boldsymbol{e}) \cdot \prod_{i=1}^{\mu} \big(e_i \cdot x_i + (1 - e_i)(1 - x_i)\big) \\ &= \sum_{\boldsymbol{e} \in \{0,1\}^\mu} f(\boldsymbol{e}) \cdot eq(\boldsymbol{x}, \boldsymbol{e}).\end{aligned}$$

Here, the function $eq(\boldsymbol{x}, \boldsymbol{e})$ is defined as the multilinear extension of the Boolean equality function $eq(\boldsymbol{x}, \boldsymbol{e})$. In the Boolean space, the equality function $eq(\boldsymbol{x}, \boldsymbol{e})$ evaluates to:

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

The value $\widetilde{f}(\boldsymbol{r})$ for any $\boldsymbol{r} \in \mathbb{F}^{\mu}$ can be computed in $O(2^{\mu})$ operations over $\mathbb{F}$.

2.3 The Logarithmic Derivative

The logarithmic derivative of a polynomial $p(X)$ over a field $\mathbb{F}$ is defined as the rational function $p'(X)/p(X)$. Crucially, applying the logarithmic derivative to a product of linear factors transforms it into a sum of simple fractions. This algebraic property allows us to elegantly reduce multiset equality checks to rational function identities.

Lemma 2 (Multiset Check). *The condition for two multisets $\{a_i\}_1^n$, $\{b_i\}_1^n$ $(a_i, b_i \in \mathbb{F}_p)$ to be equal is that $\prod_{i=1}^n (X + a_i) = \prod_{i=1}^n (X + b_i)$ holds in $\mathbb{F}(X)$, which is equivalent to:*

$$\sum_{i=1}^{n} \frac{1}{X + a_i} = \sum_{i=1}^{n} \frac{1}{X + b_i}.$$

The proof is given in [18].

This lemma is often used to construct lookup arguments [12]. The goal is for the prover to convince the verifier that, given committed polynomials $f(x)$ and $t(x)$, the set $f(\mathbb{D}) := \{f(u) : u \in \mathbb{D}\}$ is a subset of $t(\mathbb{H}) := \{t(h) : h \in \mathbb{H}\}$, where $\mathbb{D}$ and $\mathbb{H}$ are domains in $\mathbb{F}$. Here, $f(\mathbb{D})$ is the lookup vector, and $t(\mathbb{H})$ is the table, with $m := |\mathbb{D}|$ and $N := |\mathbb{H}|$. To prove $f(\mathbb{D}) \subseteq t(\mathbb{H})$, the prover relies on the identity:

$$\sum_{u \in \mathbb{D}} \frac{1}{x + f(u)} = \sum_{h \in \mathbb{H}} \frac{m_h}{x + t(h)},$$

where m_h represents the multiplicity of $t(h)$ in $f(\mathbb{D})$, i.e., the number of times $t(h)$ appears in $f(u)$.

2.4 Argument of Knowledge

An argument of knowledge Π is a protocol involving two parties, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. In general, it consists of three algorithms, namely Setup, Prove and Verify, and allows the prover to convince the verifier that given a string x it knows a witness w such that (x, w) is in an NP relation.

Definition 3. *An argument of knowledge for an indexed family of NP relations $\{\mathcal{R}_{ind}\}_{ind \in \mathcal{I}}$ is a triple of algorithms $\Pi = (\text{Setup}, \mathcal{P}, \mathcal{V})$ with the following syntax:*

- $(\text{pp}, \text{vp}) \leftarrow \text{Setup}\left(ind, 1^{\lambda}\right)$*: in the offline phase,* Setup *is given the index ind and 1^{λ} outputs public parameters* pp*, and verifier public parameters* vp*.*

- $b \leftarrow \langle \mathcal{P}(\mathrm{pp}, x, w), \mathcal{V}(\mathrm{vp}, x) \rangle$*: in the online phase,* $\mathcal{P}$ *receives input* pp *and a pair of* $(x, w) \in \mathcal{R}_{ind}$*, and* $\mathcal{V}$ *receives input* vp *and* x*. The parties interact with each other. Finally,* $\mathcal{V}$ *outputs 0 or 1.*

The algorithms should satisfy the following security requirements.

- *Completeness. For any* $ind \in \mathcal{I}$ *and* $(x, w) \in \mathcal{R}_{ind}$*,*

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

- *Knowledge soundness. For any PPT algorithm* $\mathcal{P}^*$*, there exists a PPT extractor* E *such that for any* $ind \in \mathcal{I}$ *and auxiliary inputs* $z \in \{0,1\}^*$*,*

$$\Pr\left[b = 1 \wedge (x, w) \notin \mathcal{R}_{ind} \;\middle|\; \begin{array}{c} (\mathrm{pp}, \mathrm{vp}) \leftarrow \mathrm{Setup}\left(ind, 1^{\lambda}\right) \\ (x, \mathrm{st}) \leftarrow \mathcal{P}^*(\mathrm{pp}, z, \bot) \\ b \leftarrow \langle \mathcal{P}^*(\mathrm{pp}, z, \mathrm{st}; r), \mathcal{V}(\mathrm{vp}, x) \rangle \\ w \leftarrow \mathrm{E}(\mathrm{pp}, z; r) \end{array}\right] = \; negl\,.$$

SNARK. SNARKs can be constructed from information-theoretic proof systems that provide the verifier with oracle access to prover messages, such as Polynomial Interactive Oracle Proof (PIOP). The information-theoretic proof is then compiled using cryptographic tools, such as a polynomial commitment scheme.

2.5 Multilinear Polynomial Commitment

Definition 4 (Commitment scheme). *A commitment scheme* Γ *is a tuple* $\Gamma = (\mathrm{Setup}, \mathrm{Commit}, \mathrm{Open})$ *of PPT algorithms where:*

- $\mathrm{gp} \leftarrow \mathrm{Setup}(1^{\lambda})$ *generates public parameters* gp*;*
- $(C, r) \leftarrow \mathrm{Commit}(\mathrm{gp}, x)$ *takes a secret message* x *and outputs a public commitment* C *and (optionally) a secret opening hint* r *(which might or might not be the randomness used in the computation);*
- $b \in \{0,1\} \leftarrow \mathrm{Open}(\mathrm{gp}, C, x, r)$ *verifies the opening of commitment* C *to the message* x *provided with the opening hint* r*.*

Definition 5 (Multilinear Polynomial Commitment). *A polynomial commitment scheme is a tuple of protocols* $\Gamma = (\mathrm{Setup}, \mathrm{Commit}, \mathrm{Open}, \mathrm{Eval})$ *where* (Setup, Commit, Open) *is a binding commitment scheme for a message space* $R[X]$ *of polynomials over some ring* R*, and*

- $b \in \{0,1\} \leftarrow \mathrm{Eval}((\mathrm{vp}, \mathrm{pp}), C, \mathbf{z}, y, d, \mu; f)$ *is an interactive public-coin protocol between a PPT prover* $\mathcal{P}$ *and verifier* $\mathcal{V}$*. Both* $\mathcal{P}$ *and* $\mathcal{V}$ *have as input a commitment* C*, points* $\mathbf{z} \in \mathbb{F}^{\mu}$ *and* $y \in \mathbb{F}$*. The prover has prover parameters* pp*, and the verifier has verifier parameters* vp*. The prover additionally knows the opening of* C *to a secret multilinear polynomial* $f \in \mathcal{F}_{\mu}$*. The protocol convinces the verifier that* $f(\mathbf{z}) = y$*.*

A polynomial commitment scheme is correct if an honest committer can successfully convince the verifier of any evaluation. Specifically, if the prover is honest, then for all polynomials $f \in \mathcal{F}_\mu$ and all points $z \in \mathbb{F}^\mu$,

$$\Pr\left[b = 1 \;\middle|\; \begin{array}{l} \mathrm{gp} \leftarrow \mathrm{Setup}\left(1^\lambda, \mu\right) \\ (C, r) \leftarrow \mathrm{Commit}(\mathrm{gp}, f) \\ y \leftarrow f(\mathbf{z}) \\ b \leftarrow \mathrm{Eval}(\mathrm{gp}, c, z, y, d, \mu; f, r) \end{array}\right] = 1.$$

2.6 PIOP Compilation

PIOP compilation transforms the interactive oracle proof into an interactive argument of knowledge (without oracles) Π. The compilation replaces the oracles with polynomial commitments. Every query by the verifier is replaced with an invocation of the Eval protocol at the query point $\boldsymbol{z}$. The compiled verifier accepts if the PIOP verifier accepts and if the output of all Eval invocations is 1. If Π is public-coin, then it can further be compiled to a non-interactive argument of knowledge (or NARK) using the Fiat-Shamir transform.

Theorem 1 (PIOP Compilation). *If the polynomial commitment scheme Γ has witness-extended emulation, and if the t-round Polynomial IOP for $\mathcal{R}$ has negligible knowledge error, then Π, the output of the PIOP compilation, is a secure (non-oracle) argument of knowledge for $\mathcal{R}$. The compilation also preserves zero knowledge. If Γ is hiding and Eval is honest-verifier zero-knowledge, then Π is honest-verifier zero-knowledge. The efficiency of the resulting argument of knowledge Π depends on the efficiency of both the PIOP and Γ :*

- *Prover time The prover time is equal to the sum of (i) prover time of the PIOP, (ii) the oracle length times the commitment time, and (iii) the query complexity times the prover time of Γ.*
- *Verifier time The verifier time is equal to the sum of (i) the verifier time of the PIOP and (ii) the verifier time for Γ times the query complexity of the PIOP.*
- *Proof size The proof size is equal to the sum of (i) the message complexity of the PIOP times the commitment size and (ii) the query complexity times the proof size of Γ. If the proof size is $O\left(\log^c(|\boldsymbol{w}|)\right)$, then we say the proof is succinct.*

2.7 Sum-Check

[19] introduced the sum-check protocol, which allows a verifier $\mathcal{V}$ to delegate the computation of $v = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} f(\boldsymbol{b})$ to an unbounded prover $\mathcal{P}$, who then establishes the correctness of the sum.

Definition 6 (Sum-check relation). *The relation $\mathcal{R}_{SUM}$ is the set of tuples $(x; \omega) = ((v, [[f]]); f)$, where f has u independent variables and a degree of d, and $\sum_{\boldsymbol{b} \in \{0,1\}^\mu} f(\boldsymbol{b}) = v$.*

Construction. Given a tuple $(v, [[f]]; f)$, the sum-check is defined as follows.

- For $i = \mu, \mu - 1, \ldots, 1$:
  - The prover computes $r_i(x_i) := \sum_{\boldsymbol{b} \in \{0,1\}^{i-1}} f(\boldsymbol{b}, x_i, \alpha_i, \ldots, \alpha_\mu)$ and sends r_i to the verifier. r_i is a univariate polynomial of degree at most d.
  - The verifier checks that $v = r_i(0) + r_i(1)$, samples $\alpha_i \leftarrow \mathbb{F}$, sends α_i to the prover, and sets $v \leftarrow r_i(\alpha_i)$.
- Finally, the verifier accepts if $f(\alpha_1, \ldots, \alpha_\mu) = v$.

Theorem 2 (Sum-check PIOP [9]). *The PIOP for $\mathcal{R}_{SUM}$ is perfectly complete and has knowledge error $\delta_{sum}^{d,\mu} := d\mu/|\mathbb{F}|$.*

We refer to [28] for the proof of the theorem.

Batching. Multiple sum-check instances, such as $(s, [[f]])$ and $(s', [[g]])$, can be efficiently batched together. This is achieved through a random-linear combination, demonstrating that $(s + \alpha s', [[f]] + \alpha[[g]]) \in \mathcal{L}(\mathcal{R}_{\text{SUM}})$, where α is a random challenge sent by verifier. The batching step has soundness $1/|\mathbb{F}|$.

Complexity. The verifier and communication costs amount to $O(d\mu)$. The prover time is $O(2^\mu \cdot d \log^2 d)$, primarily due to the application of fast polynomial multiplication techniques, as elaborated in [9].

2.8 GKR Protocol

We refer to Libra [29] for background of the GKR protocol [15]. Before describing the GKR protocol, we introduce some additional notations.

Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a layered arithmetic circuit with depth d. Without loss of generality, we assume n and k are powers of 2, padding the layers with dummy gates if necessary. Let S_i denote the number of gates in the i-th layer, and let $s_i = \lceil \log_2 S_i \rceil$. For simplicity, we assume S_i is a power of 2. We define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that maps a binary string $\boldsymbol{b} \in \{0,1\}^{s_i}$ to the output of gate $\boldsymbol{b}$ in layer i, where $\boldsymbol{b}$ is the gate label. Specifically, V_0 represents the circuit's output, while V_d corresponds to the input layer.

We define two Boolean functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$, known as wiring predicates. These functions take a gate label $\boldsymbol{z} \in \{0,1\}^{s_{i-1}}$ from layer $i-1$ and two gate labels $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^{s_i}$ from layer i, and output 1 if and only if gate $\boldsymbol{z}$ is an addition (for add_i) or multiplication (for $mult_i$) gate with inputs from gates $\boldsymbol{x}$ and $\boldsymbol{y}$. Using these predicates, the value $V_i(\boldsymbol{z})$ can be expressed recursively as:

$$V_i(\boldsymbol{z}) = \sum_{\boldsymbol{x},\boldsymbol{y} \in \{0,1\}^{s_{i+1}}} add_{i+1}(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y})\big(V_{i+1}(\boldsymbol{x}) + V_{i+1}(\boldsymbol{y})\big) + mult_{i+1}(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y})\big(V_{i+1}(\boldsymbol{x}) V_{i+1}(\boldsymbol{y})\big).$$

To apply the sum-check protocol, we utilize multilinear extensions (MLE). Let $\widetilde{V}$, $\widetilde{add}$, and $\widetilde{mult}$ be the MLEs of their respective functions over $\mathbb{F}$. The

recursive relation is rewritten as:

$$\widetilde{V}_i(\boldsymbol{z}) = \sum_{\boldsymbol{x},\boldsymbol{y}\in\{0,1\}^{s_{i+1}}} \widetilde{add}_{i+1}(\boldsymbol{z},\boldsymbol{x},\boldsymbol{y})\big(\widetilde{V}_{i+1}(\boldsymbol{x}) + \widetilde{V}_{i+1}(\boldsymbol{y})\big) + \widetilde{mult}_{i+1}(\boldsymbol{z},\boldsymbol{x},\boldsymbol{y})\big(\widetilde{V}_{i+1}(\boldsymbol{x})\widetilde{V}_{i+1}(\boldsymbol{y})\big).$$

By iteratively invoking the sum-check protocol for each layer $i = 1, \ldots, d$, the verifier reduces a claim regarding the circuit's output $\widetilde{V}_0$ to a single evaluation of the input $\widetilde{V}_d$ at a random point.

Theorem 3. *[29] Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a depth-d layered arithmetic circuit. GKR protocol is an interactive proof for the function computed by C, achieving soundness $O(d \log |C|/|\mathbb{F}|)$. It employs $O(d \log |C|)$ rounds of interaction, and the running time of the prover $\mathcal{P}$ is $O(|C|)$. Let the optimal computation time for all $\widetilde{add}_i$ and $\widetilde{mult}_i$ operations be T; then the running time of $\mathcal{V}$ is $O(n + k + d \log |C| + T)$. Specifically, for log-space uniform circuits, T is polylogarithmic in $|C|$, i.e., $T = \text{polylog}\,|C|$.*

2.9 Customizable Constraint System

Customizable Constraint System (CCS) [24] generalizes Rank-1 Constraint Systems (R1CS) [23] without additional overhead as follows.

Definition 7. *A Customizable Constraint System is defined as follows:*

- *Size bounds $m, n, \ell, t, q, d \in \mathbb{N}$ with $n > \ell$;*
- *A sequence of matrices $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times m}$, with at most $n = \Omega(m)$ non-zero entries in total;*
- *A sequence of q multisets $\{S_0, \ldots, S_{q-1}\}$, where each multiset has elements from $\{0, \ldots, t-1\}$ and cardinality at most d;*
- *A sequence of q constants $\{c_0, \ldots, c_{q-1}\}$, where each constant is from $\mathbb{F}$.*
- *A CCS instance consists of a public input $\boldsymbol{x} \in \mathbb{F}^\ell$.*
- *A CCS witness consists of a vector $\boldsymbol{w} \in \mathbb{F}^{n-\ell-1}$.*
- *A CCS structure-instance tuple $(S, \boldsymbol{x})$ is satisfied by a CCS witness $\boldsymbol{w}$ if: $\sum_{i=0}^{q-1} c_i \cdot (\bigcirc_{j \in S_i} M_j \cdot \boldsymbol{z}) = \boldsymbol{0}$, where $\boldsymbol{z} = (\boldsymbol{w}, 1, \boldsymbol{x}) \in \mathbb{F}^m$, $M_j \cdot \boldsymbol{z}$ denotes matrix-vector multiplication, $\bigcirc$ denotes the Hadamard product between vectors, $\boldsymbol{0}$ is an m-dimensional vector with all entries equal to the additive identity in $\mathbb{F}$.*

3 An Efficient Sum-Check for High-Degree Polynomials

In this section, we consider the scenario where a prover demonstrates to a verifier that the sum of a multivariate polynomial $f(\boldsymbol{x})$ over all $\boldsymbol{x} \in \{0,1\}^\mu$ equals a specific value v, where $f(\boldsymbol{x})$ is represented as

$$f(\boldsymbol{x}) := h\big(g_1(\boldsymbol{x}), \ldots, g_c(\boldsymbol{x})\big).$$

Here $g_1(\boldsymbol{x})$ is a multilinear polynomial. The polynomial h, which has degree d, can be computed by a general arithmetic circuit of depth $O(\log d)$, commonly referred to as a low-depth circuit.

In previous approaches, the sum-check protocol (in Subsection 2.7) is directly employed for the high-degree polynomial. Specifically, in each round, the prover computes a universal polynomial $r_i(x_i)$. As detailed in [27,29], the computation of $r_i(x_i)$ involves two main steps: first, evaluating the function h at d distinct points, followed by performing interpolation to derive the polynomial $r_i(x_i)$. This approach exhibited a time complexity of $O(2^\mu d^2)$. In [9], the authors introduced an optimized method that reduces the complexity to $O(2^\mu \cdot d \log^2 d)$ for low-depth circuits, such as $f(\boldsymbol{x}) := \prod_c g_c(\boldsymbol{x})$. This improvement is achieved by using fast polynomial multiplication.

We optimize the prover runtime to $O(2^\mu \cdot d)$. This is done by embedding the generalized GKR protocol [31] into the sum-check protocol for high-degree polynomials. Specifically, we define $\widetilde{f}(\boldsymbol{x})$ implicitly as the multilinear extension of f based on its evaluations over the Boolean hypercube:

$$\widetilde{f}(\boldsymbol{x}) = \sum_{\boldsymbol{b}\in\{0,1\}^\mu} f(\boldsymbol{b}) \cdot eq(\boldsymbol{x}, \boldsymbol{b}) = \sum_{\boldsymbol{b}\in\{0,1\}^\mu} h\big(g_1(\boldsymbol{b}), \ldots, g_c(\boldsymbol{b})\big) \cdot eq(\boldsymbol{x}, \boldsymbol{b}).$$

By the standard properties of multilinear extensions (Definition 2), the target sum is preserved:

$$v = \sum_{\boldsymbol{b}\in\{0,1\}^\mu} f(\boldsymbol{b}) = \sum_{\boldsymbol{b}\in\{0,1\}^\mu} \widetilde{f}(\boldsymbol{b}).$$

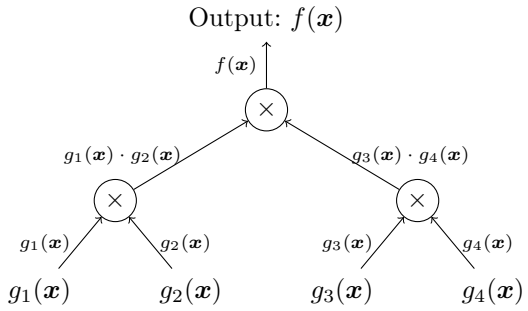
To compute this sum efficiently without explicitly expanding the high-degree composite polynomial f, our protocol proceeds in two phases:

1. **Phase 1: Sum-check for v.** The prover and verifier execute a standard sum-check protocol on the low-degree multilinear polynomial $\widetilde{f}$ to verify the claim $\sum_{\boldsymbol{b}\in\{0,1\}^\mu} \widetilde{f}(\boldsymbol{b}) = v$. This phase reduces the global sum over the hypercube to a single claimed evaluation $\widetilde{f}(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha} \in \mathbb{F}^\mu$ is a random challenge chosen by the verifier $\mathcal{V}$.
2. **Phase 2: Generalized GKR for $\widetilde{f}(\boldsymbol{\alpha})$.** Instead of evaluating $\widetilde{f}(\boldsymbol{\alpha})$ directly—which would require expanding h—we validate it using the generalized GKR protocol. Conceptually, the 2^μ evaluations of $f(\boldsymbol{b})$ correspond precisely to the outputs of a data-parallel circuit C comprising 2^μ independent copies of the subcircuit h. Consequently, $\widetilde{f}$ is exactly the multilinear extension of this circuit's output layer. The verification is thus elegantly delegated to the GKR protocol, where the input layer of C is encoded as:

$$\widetilde{V}_{\text{in}}(\boldsymbol{z}_1, \boldsymbol{z}_2) = \sum_{i\in[c]} eq(\boldsymbol{z}_1, \langle i \rangle) \cdot g_i(\boldsymbol{z}_2).$$

   Here, $\boldsymbol{z}_1 \in \{0,1\}^{\lceil \log c \rceil}$ indexes the specific input polynomial g_i, and $\boldsymbol{z}_2 \in \{0,1\}^\mu$ specifies the evaluation point over the Boolean hypercube. Querying the oracles $[[g_i]]$ at this input layer inherently enforces the correct polynomial composition.

Next, we detail the implementation of the second-phase validation. For instance, consider $f(\boldsymbol{x}) := h\big(g_1(\boldsymbol{x}), \ldots, g_c(\boldsymbol{x})\big) = \prod_{k=1}^{4} g_k(\boldsymbol{x})$. The computation of $f(\boldsymbol{x})$ is compiled into a two-layer subcircuit structured as follows:



The circuit C comprises 2^{μ} identical subcircuits h arranged in parallel, with its input structured as 2^{μ} distinct tuples. Each tuple corresponds to the evaluations of polynomials g_1, g_2, g_3, g_4 at a specific point $\boldsymbol{x} \in \{0,1\}^{\mu}$, denoted as $(g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), g_3(\boldsymbol{x}), g_4(\boldsymbol{x}))$. To formally capture the input structure, we define the multilinear extension of the input layer as

$$\widetilde{V}_2(\boldsymbol{z}_1, \boldsymbol{z}_2) := \sum_{i \in [c]} eq(\boldsymbol{z}_1, \langle i \rangle) \cdot g_i(\boldsymbol{z}_2),$$

where $\boldsymbol{z}_1 \in \{0,1\}^{\lceil \log c \rceil}$ and $\boldsymbol{z}_2 \in \{0,1\}^{\mu}$. One can verify that $\widetilde{V}_2(\langle i \rangle, \boldsymbol{x}) = g_i(\boldsymbol{x})$ holds as intended. According to the GKR protocol in Subsection 2.8, the value of each gate at layer i can be expressed as a function of the values at layer $i+1$ through the following relation:

$$\begin{aligned}\widetilde{V}_i(\boldsymbol{z}) = \sum_{\boldsymbol{x},\boldsymbol{y} \in \{0,1\}^{s_{i+1}}} & \widetilde{\mathrm{add}}_{i+1}(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y}) \left(\widetilde{V}_{i+1}(\boldsymbol{x}) + \widetilde{V}_{i+1}(\boldsymbol{y})\right) \\ & + \widetilde{\mathrm{mult}}_{i+1}(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y}) \left(\widetilde{V}_{i+1}(\boldsymbol{x}) \widetilde{V}_{i+1}(\boldsymbol{y})\right).\end{aligned}$$

This requires the verifier to evaluate both $\widetilde{\mathrm{add}}_{i+1}$ and $\widetilde{\mathrm{mult}}_{i+1}$ in the GKR protocol. To perform this computation efficiently, we next leverage the data-parallel structure of circuit C to represent $\widetilde{V}_i(\boldsymbol{z})$.

Let $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2) \in \{0,1\}^{\mu} \times \{0,1\}^{k_i}$ be the label of a gate at layer i, where $\boldsymbol{z}_1$ indicates which sub-circuit of C the gate belongs to, and $\boldsymbol{z}_2$ specifies the label of the gate within that sub-circuit. Similarly, let $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2) \in \{0,1\}^{\mu} \times \{0,1\}^{k_{i+1}}$ and $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2) \in \{0,1\}^{\mu} \times \{0,1\}^{k_{i+1}}$ be the labels of two gates at layer $i+1$. For any $\boldsymbol{z} \in \{0,1\}^{\mu+k_i}$, the function V_i satisfies $V_i(\boldsymbol{z}_1, \boldsymbol{z}_2) = W_i(\boldsymbol{z}_1, \boldsymbol{z}_2)$, with W_i defined by:

$$\begin{aligned}W_i(\boldsymbol{z}_1, \boldsymbol{z}_2) = \sum_{\boldsymbol{x}_2,\boldsymbol{y}_2 \in \{0,1\}^{k_{i+1}}} & \widetilde{\mathrm{add}}_{i+1}(\boldsymbol{z}_2, \boldsymbol{x}_2, \boldsymbol{y}_2) \left(\widetilde{V}_{i+1}(\boldsymbol{z}_1, \boldsymbol{x}_2) + \widetilde{V}_{i+1}(\boldsymbol{z}_1, \boldsymbol{y}_2)\right) \\ & + \widetilde{\mathrm{mult}}_{i+1}(\boldsymbol{z}_2, \boldsymbol{x}_2, \boldsymbol{y}_2) \left(\widetilde{V}_{i+1}(\boldsymbol{z}_1, \boldsymbol{x}_2) \widetilde{V}_{i+1}(\boldsymbol{z}_1, \boldsymbol{y}_2)\right).\end{aligned}$$

This states that an addition (respectively, multiplication) gate $\boldsymbol{z} \in \{0,1\}^{\mu+k_i}$ of C is connected to gates $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^{\mu+k_{i+1}}$ of C if and only if $\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y}$ all belong to the same sub-circuit of C (i.e., $\boldsymbol{z}_1 = \boldsymbol{x}_1 = \boldsymbol{y}_1$), and within that sub-circuit, gate $\boldsymbol{z}$ is connected to gates $\boldsymbol{x}$ and $\boldsymbol{y}$.

Finally, the polynomial $W_i(\boldsymbol{z}_1, \boldsymbol{z}_2)$ is of degree 2 in $\boldsymbol{z}_1$ and degree 1 in $\boldsymbol{z}_2$. Consequently, the (unique) multilinear extension $\widetilde{V}_i$ of V_i is:

$$\widetilde{V}_i(\boldsymbol{z}_1, \boldsymbol{z}_2) = \sum_{\boldsymbol{e} \in \{0,1\}^{\mu}} eq(\boldsymbol{z}_1, \boldsymbol{e}) \cdot W_i(\boldsymbol{e}, \boldsymbol{z}_2).$$

For a general circuit h, we refer to the generalized GKR protocol [31] with linear prover time. In this framework, gates at layer i may receive inputs from gates across multiple subsequent layers, specifically from layer $i+1$ through layer d. To accommodate this extended layered dependency, a key assumption is introduced: each gate at layer i must take at least one input directly from the immediately subsequent layer $i+1$, which is designated as its left operand. Under this assumption, the multilinear extension $\widetilde{V}_i$ of the circuit values at layer i can be reformulated in terms of values from deeper layers as follows:

$$\begin{aligned}\widetilde{V}_i(\boldsymbol{z}) = \sum_{j=i+1}^{d} \sum_{\substack{\boldsymbol{y}_j \in \{0,1\}^{s_j} \\ \boldsymbol{x}_{i+1} \in \{0,1\}^{s_{i+1}}}} & \widetilde{\mathrm{add}}_{i+1,j}(\boldsymbol{z}, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j)\big(\widetilde{V}_{i+1}(\boldsymbol{x}_{i+1}) + \widetilde{V}_j(\boldsymbol{y}_j)\big) \\ & + \widetilde{\mathrm{mult}}_{i+1,j}(\boldsymbol{z}, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j)\big(\widetilde{V}_{i+1}(\boldsymbol{x}_{i+1})\widetilde{V}_j(\boldsymbol{y}_j)\big),\end{aligned}$$

where $\widetilde{\mathrm{add}}_{i+1,j}(\boldsymbol{z}_i, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j) = 1$ ($\widetilde{\mathrm{mult}}_{i+1,j}(\boldsymbol{z}_i, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j) = 1$) if and only if gate $\boldsymbol{z}_i$ in layer $i+1$ is the addition (multiplication) of values $\widetilde{V}_{i+1}(\boldsymbol{x}_{i+1})$ and $\widetilde{V}_j(\boldsymbol{y}_j)$.

Building on the data-parallel structure of circuit C, we now express $\widetilde{V}_i(\boldsymbol{z})$ in a similar manner. For $\boldsymbol{z} \in \{0,1\}^{\mu+k_i}$, we have $V_i(\boldsymbol{z}_1, \boldsymbol{z}_2) = W_i(\boldsymbol{z}_1, \boldsymbol{z}_2)$:

$$\begin{aligned}W_i(\boldsymbol{z}_1, \boldsymbol{z}_2) = \sum_{j=i+1}^{d} \sum_{\substack{\boldsymbol{y}_j \in \{0,1\}^{k_j} \\ \boldsymbol{x}_{i+1} \in \{0,1\}^{k_{i+1}}}} & \widetilde{\mathrm{add}}_{i+1,j}(\boldsymbol{z}_2, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j)\big(\widetilde{V}_{i+1}(\boldsymbol{z}_1, \boldsymbol{x}_{i+1}) + \widetilde{V}_j(\boldsymbol{z}_1, \boldsymbol{y}_j)\big) \\ & + \widetilde{\mathrm{mult}}_{i+1,j}(\boldsymbol{z}_2, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j)\big(\widetilde{V}_{i+1}(\boldsymbol{z}_1, \boldsymbol{x}_{i+1})\widetilde{V}_j(\boldsymbol{z}_1, \boldsymbol{y}_j)\big),\end{aligned}$$

From this, we conclude that the multilinear extension $\widetilde{V}_i$ of V_i is given by:

$$\widetilde{V}_i(\boldsymbol{z}_1, \boldsymbol{z}_2) = \sum_{\boldsymbol{e} \in \{0,1\}^{\mu}} eq(\boldsymbol{z}_1, \boldsymbol{e}) \cdot W_i(\boldsymbol{e}, \boldsymbol{z}_2).$$

Prover Cost. The key to achieving an $O(|h| \cdot 2^{\mu})$ runtime for an honest prover builds upon the GKR protocol with linear prover time.

Verifier Cost. To bound the verifier time, it is necessary to evaluate $\widetilde{\mathrm{add}}$ and $\widetilde{\mathrm{mult}}$ at random points. When a linear prover time generalized GKR protocol is applied to the subcircuit h, the prover can perform these evaluations in $O(|h|)$

time. Consequently, the verifier can also evaluate $\widetilde{\text{add}}$ and $\widetilde{\text{mult}}$ in $O(|h|)$ time, which remains independent of the number of subcircuit copies 2^μ.

Additionally, the verifier must evaluate the eq function at a random point in each layer i. Since this can be done using $O(\mu)$ field operations for any input, it does not affect the overall asymptotic time complexity.

Definition 8 (Doubly efficient sum-check (DESC) relation). *The relation $\mathcal{R}_{DESC}$ is defined as the set of tuples $(i; x; \omega) = \left(h; \left(v, \left[\left[\widetilde{V}_{in}\right]\right]\right); \widetilde{V}_{in}\right)$, where the input multilinear polynomial oracle $\widetilde{V}_{in}$ is given by*

$$\left[\left[\widetilde{V}_{in}(\boldsymbol{z})\right]\right] = \sum_{i\in[c]} eq(\langle i\rangle, \boldsymbol{z}_1) \cdot [[g_i(\boldsymbol{z}_2)]]\,.$$

Here, $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$, $\boldsymbol{z}_1 \in \mathbb{F}^{\lceil \log c\rceil}$, and $\boldsymbol{z}_2 \in \mathbb{F}^\mu$. Each g_i is a multilinear polynomial. The function h is computable by a general arithmetic circuit of depth $O(\log d)$, and it satisfies $\sum_{\boldsymbol{x}\in\{0,1\}^\mu} h\big(g_1(\boldsymbol{x}), \ldots, g_c(\boldsymbol{x})\big) = v$.

Construction. Given a tuple $\left(h; \left(v, \{[[g_i]]\}_{i\in[c]}\right); \{g_i\}_{i\in[c]}\right)$ and a μ-variate polynomial f such that $f(\boldsymbol{x}) = h\big(g_1(\boldsymbol{x}), \ldots, g_c(\boldsymbol{x})\big)$, the DESC protocol is described as follows:

- Run a sum-check to convince $\mathcal{V}$ that $\left(\left(v, [[\widetilde{f}]]\right); \widetilde{f}\right) \in \mathcal{R}_{\text{SUM}}$, where in each round, the prover sends a univariate polynomial r_i of degree ≤ 1.
- The verifier $\mathcal{V}$ receives $\widetilde{f}(\alpha_1, \ldots, \alpha_\mu) = v$ and runs a generalized GKR for:

$$\left(C; \left(v, \left[\left[\widetilde{V}_{in}\right]\right]\right); \left(\widetilde{V}_{in}\right)\right) \in \mathcal{R}_{\text{GKR}}.$$

- Finally, $\mathcal{V}$ queries oracles $\{[[g_i]]\}_{i\in[c]}$ to verify the claim of the input $\widetilde{V}_{in}$.

Theorem 4 (DESC PIOP). *The PIOP for $\mathcal{R}_{DESC}$ is perfectly complete and has knowledge error $\delta^{d,\mu}_{desc} := O\big(\log d \cdot (\mu + \log|h|)/|\mathbb{F}|\big)$.*

Proof. **Completeness error:** The completeness is straightforward by the completeness of the sum-check and the generalized GKR.

Knowledge Soundness: The first sum-check protocol exhibits knowledge error of $O(\mu/|\mathbb{F}|)$. Regarding the evaluation verification phase, Theorem 2 of [31] states that the generalized GKR protocol has a soundness error of $O(D \cdot \log S/|\mathbb{F}|)$, where D is the circuit depth and S is the circuit width. In our construction, the verification is performed over a data-parallel circuit consisting of 2^μ copies of the sub-circuit h, which has a depth of $D = O(\log d)$ and a width of $S = |h| \cdot 2^\mu$. Substituting these parameters, the error for this phase is $O(\log d \cdot (\mu + \log|h|)/|\mathbb{F}|)$. Consequently, the overall soundness error is dominated by the second phase and bounded by $O\left(\log d \cdot (\mu + \log|h|)/|\mathbb{F}|\right)$.

Complexity. Finally, building upon the efficiency results of the generalized GKR protocol, we summarize the complexity analysis. Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a general arithmetic circuit of depth D. The generalized GKR protocol (Protocol 3) achieves a prover time of $O(|C|)$, a proof size of $O(D \log |C| + D^2)$, and a verifier time of $O(D \log |C| + D^2 + T)$, where T represent the costs to evaluate the multilinear extensions of the wiring predicates at random points.

For the DESC PIOP, the verification circuit is structured as 2^μ parallel copies of a subcircuit h with depth $O(\log d)$. By substituting the circuit size $|C| = 2^\mu |h|$ and depth $D = O(\log d)$ into the general bounds, and leveraging the structural uniformity for efficient verifier evaluations, we obtain the following complexity:

- The prover time is $\mathrm{tp}^h_{\mathrm{desc}} = O\left(|h| \cdot 2^\mu\right)$.
- The verifier time is $\mathrm{tv}^h_{\mathrm{desc}} = O\big(|h| + \log^2 d + \log d \cdot (\mu + \log |h|)\big)$.
- The total proof size is $\mathrm{ps}^h_{\mathrm{desc}} = O\big(\log^2 d + \log d \cdot (\mu + \log |h|)\big)$.
- The query complexity is $\mathrm{q}^h_{\mathrm{desc}} = c$, with proof oracles $\{g_i\}_{i \in [c]}$.

4 CCS with Linear Prover Time

This section presents a PIOP for CCS (detailed in Subsection 2.9) that achieves linear prover time. A CCS structure-instance pair $(S, \boldsymbol{x})$ is said to be satisfiable if there exists a witness vector $\boldsymbol{w}$ such that the following condition holds:

$$\sum_{i=0}^{q-1} c_i \cdot \left(\bigcirc_{j \in S_i} M_j \cdot \boldsymbol{z}\right) = \boldsymbol{0}.$$

Here, q is the number of multisets S_i, each satisfying $S_i \subset [t]$ and $|S_i| \leq d$; each c_i is a constant in $\mathbb{F}$; $\boldsymbol{z} = (\boldsymbol{w}, 1, \boldsymbol{x})$ is a witness-instance pair; and $M_0, \ldots, M_{t-1} \in \mathbb{F}^{m \times m}$ are matrices with at most $n = \Omega(m)$ non-zero entries in total. The protocol is initiated by running a sum-check protocol to check:

$$\sum_{\boldsymbol{x} \in \{0,1\}^{\log m}} eq(\boldsymbol{\alpha}, \boldsymbol{x}) f(\boldsymbol{x}) = 0, \tag{1}$$

where $\boldsymbol{\alpha}$ is a given random challenge vector, and the function $f(\boldsymbol{x})$ is defined as:

$$f(\boldsymbol{x}) := h\big(g_1(\boldsymbol{x}), \ldots, g_t(\boldsymbol{x})\big) = \sum_{i=0}^{q-1} c_i \cdot \prod_{j \in S_i} g_j(\boldsymbol{x}),$$

$$g_j(\boldsymbol{x}) := \sum_{\boldsymbol{y} \in \{0,1\}^{\log m}} \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \widetilde{\boldsymbol{z}}(\boldsymbol{y}).$$

Directly applying the sum-check protocol to equation (1) results in a prover time of $O(qmd \log^2 d)$. This complexity arises primarily from using FFTs to multiply d degree-1 polynomials. Although d is typically small (e.g., 2 or 5), it may take larger values in practice, potentially exceeding 100 in some instances.

To address the prover time in this setting, we design a PIOP for CCS that achieves a complexity of $O(qmd)$. Specifically, we reformulate the computation of h as a general subcircuit of size $O(qd)$. The circuit C is composed of m such subcircuits, and the multilinear extension of its input is defined as:

$$\widetilde{V}_{\text{in}}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{j \in [t]} \widetilde{\mathsf{eq}}(\boldsymbol{x}_1, \langle j \rangle) \cdot \sum_{\boldsymbol{y} \in \{0,1\}^{\log m}} \widetilde{M}_j(\boldsymbol{x}_2, \boldsymbol{y}) \cdot \widetilde{\boldsymbol{z}}(\boldsymbol{y}),$$

where $\langle j \rangle$ denotes the binary representation of j. By subsequently applying the sum-check PIOP in combination with a batched sum-check protocol, we achieve the prover complexity of $O(qmd)$.

Definition 9 (CCS Relation). *The relation $\mathcal{R}_{\mathsf{CCS}}$ is the set of all tuples*

$$\left(\left(q, \left\{ c_i, S_i, \left[\left[\widetilde{M}_i \right] \right] \right\}_{i \in [t]} \right); (\boldsymbol{x}, [[\widetilde{\boldsymbol{w}}]]); \left(\widetilde{\boldsymbol{w}}, \left\{ \widetilde{M}_i \right\}_{i \in [t]} \right) \right),$$

and satisfies the following condition. Let $\boldsymbol{z} = (\boldsymbol{w}, 1, \boldsymbol{x})$ with multilinear extension $\widetilde{\boldsymbol{z}}$, and define for each $j \in [t]$ the polynomial

$$g_j(\boldsymbol{x}) := \sum_{\boldsymbol{y} \in \{0,1\}^{\log m}} \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \widetilde{\boldsymbol{z}}(\boldsymbol{y}).$$

Then the constraint function

$$h(g_1(\boldsymbol{x}), \ldots, g_t(\boldsymbol{x})) = \sum_{i=0}^{q-1} c_i \cdot \prod_{j \in S_i} g_j(\boldsymbol{x}) = 0 \quad \textit{for all } \boldsymbol{x} \in \{0,1\}^{\log m}.$$

Construction. Given a tuple

$$\left(\left(q, \left\{ c_i, S_i, \left[\left[\widetilde{M}_i \right] \right] \right\}_{i \in [t]} \right); (\boldsymbol{x}, [[\widetilde{\boldsymbol{w}}]]); \left(\widetilde{\boldsymbol{w}}, \left\{ \widetilde{M}_i \right\}_{i \in [t]} \right) \right),$$

the CCS PIOP is described as follows:

- The verifier $\mathcal{V}$ selects a random challenge vector $\boldsymbol{\alpha}$ and sends it to prover $\mathcal{P}$.
- Let $\widetilde{f}(\boldsymbol{\alpha}) = \sum_{\boldsymbol{x} \in \{0,1\}^{\log m}} eq(\boldsymbol{\alpha}, \boldsymbol{x}) \cdot h\big(g_1(\boldsymbol{x}), \ldots, g_t(\boldsymbol{x})\big) = 0$.
- Run a generalized GKR for the relation:

  $$\left(h; \left(0, \left[\left[\widetilde{V}_{\text{in}} \right] \right] \right); \left(\widetilde{V}_{\text{in}} \right) \right) \in \mathcal{R}_{\text{GKR}},$$

  where $\left[\left[\widetilde{V}_{\text{in}}(\boldsymbol{z}) \right] \right] := \sum_{i \in [c]} \text{eq}(\langle i \rangle, \boldsymbol{z}_1) \cdot [[g_i(\boldsymbol{z}_2)]]$.
- $\mathcal{V}$ receives claims $v_i = \sum_{\boldsymbol{y} \in \{0,1\}^{\log m}} \widetilde{M}_i(\boldsymbol{\alpha}_2, \boldsymbol{y}) \cdot \widetilde{z}(\boldsymbol{y})$ for $i \in [t]$ and checks:

  $$\widetilde{V}_{\text{in}}(\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2) = \sum_{i \in [c]} \text{eq}(\langle i \rangle, \boldsymbol{\alpha}_1) \cdot g_i(\boldsymbol{\alpha}_2).$$

  If yes, $\mathcal{V}$ sends challenges $\{e_i\}_{i \in [t]}$ and runs a sum-check on:

  $$\sum_{\boldsymbol{y} \in \{0,1\}^{\log m}} \left(\sum_{i=1}^{t} e_i \cdot \widetilde{M}_i(\boldsymbol{\alpha}_2, \boldsymbol{y}) \right) \cdot \widetilde{z}(\boldsymbol{y}) = \sum_{i=1}^{t} e_i \cdot v_i.$$

- Finally, verifier $\mathcal{V}$ queries oracles $\left\{\left[\left[\widetilde{M}_i\right]\right]\right\}_{i\in[t]}$ and $[[\widetilde{\boldsymbol{w}}]]$ to validate the correctness of the last message in the sum-check protocol.

Theorem 5 (CCS PIOP). *The PIOP for $\mathcal{R}_{CCS}$ is perfectly complete and has knowledge error $\delta_{ccs}^{t,q,d,\mu} := O\Big(t + \log d \cdot \big(\mu + \log(qd)\big)\Big)/|\mathbb{F}|$.*

The proof builds upon the batched sum-check protocol and Theorem 4.

Complexity. With respect to the overhead of the DESC PIOP, we compare our protocol with the SuperSpartan PIOP for CCS [24], as detailed below.

	$\mathrm{tp}_{\mathrm{ccs}}$	$\mathrm{tv}_{\mathrm{ccs}}$	$\mathrm{ps}_{\mathrm{ccs}}$	$\mathrm{q}_{\mathrm{ccs}}$
[24]	$O\left(n + qmd\log^2 d\right)$	$O(dq + d\log m)$	$O(d\log m)$	$t+1$
This work	$O\left(n + qmd\right)$	$O\left(\begin{matrix} dq+ \\ \log d\cdot\log(qdm)\end{matrix}\right)$	$O\left(\log d\cdot\log(qdm)\right)$	$t+1$

Table 1: As detailed in the table above, the comparison assumes a CCS instance parameterized by t sparse matrices, q multisets, and a maximum algebraic degree d, subject to the bound $t \le qd$. The evaluated efficiency metrics are denoted as follows: $\mathrm{tp}_{\mathrm{ccs}}$ represents the prover time, $\mathrm{tv}_{\mathrm{ccs}}$ the verifier time, $\mathrm{ps}_{\mathrm{ccs}}$ the total proof size, and $\mathrm{q}_{\mathrm{ccs}}$ the verifier's oracle query complexity.

5 Sparse Polynomial Commitments

This section introduces an efficient PIOP designed to construct sparse polynomial commitments (SPARK). The PIOP refines existing multilinear polynomial commitment schemes, originally developed for dense polynomials, into a framework capable of efficiently managing sparse multilinear polynomials.

5.1 SPARK PIOP

The SPARK was introduced by Setty et al. [23] for the sparse polynomial $\widetilde{M}(\mathbf{x},\mathbf{y})$, where $\mathbf{x},\mathbf{y} \in \{0,1\}^{\log m}$. In this framework, let n denote the number of non-zero coefficients in $\widetilde{M}$. Without loss of generality, assume m and n are powers of two. Building on the definition of the multilinear extension provided in Definition 2, the evaluation of $\widetilde{M}(\boldsymbol{\alpha},\boldsymbol{\beta})$ can be expressed as:

$$\widetilde{M}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{\boldsymbol{x},\boldsymbol{y}\in\{0,1\}^{\log m}} M(\boldsymbol{x},\boldsymbol{y})\cdot eq(\boldsymbol{x},\boldsymbol{\alpha})\cdot eq(\boldsymbol{y},\boldsymbol{\beta}).$$

This formulation was later generalized by Setty et al. [25] to exploit the inherent tensor structure of the eq function. Specifically, they extended the original expression for $\widetilde{M}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as a sparse $2\log m$-variate multilinear polynomial to handle the evaluation of a sparse $c\log m$-variate multilinear polynomial $\widetilde{M}$ at random point $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_c)$. The generalized formulation is expressed as:

$$\begin{aligned}\widetilde{M}(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_c) &= \sum_{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_c \in \{0,1\}^{\log m}} M(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_c) \cdot \prod_{i=1}^{c} eq(\boldsymbol{x}_i, \boldsymbol{\alpha}_i) \\ &= \sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} val(\boldsymbol{k}) \cdot \prod_{i=1}^{c} E_i(\boldsymbol{k}),\end{aligned}$$

where for each $\boldsymbol{k} \in \{0,1\}^{\log n}$, the term $val(\boldsymbol{k})$ represents the $s_{id}(\boldsymbol{k})$-th nonzero entry of matrix M, and $E_i(\boldsymbol{k})$ is given by $E_i(\boldsymbol{k}) = eq(\phi_i(\boldsymbol{k}), \boldsymbol{\alpha}_i)$. Here, $\phi_i(\boldsymbol{k})$ corresponds to the i-th coordinate block $\boldsymbol{x}_i$ associated with this nonzero entry, and the index $s_{id}(\boldsymbol{k})$ is explicitly defined as the decimal representation of the binary coordinates: $s_{id}(\boldsymbol{k}) := \text{decimal}(\boldsymbol{k}) = \sum_{i=1}^{\log m} k_i \cdot 2^{i-1}$.

The "memory in the head" technique from [23] is used to verify the correctness of each E_i. In this chapter, we build upon the logarithmic derivative (in Subsection 2.3) to achieve this. Our approach further enables efficient batch verification, yielding a polynomial commitment scheme with reduced overhead. The underlying idea of this methodology is detailed below.

First, we define the index $s_{\sigma,i}(\boldsymbol{k}) = s_{id}(\boldsymbol{x}_i)$ such that the condition $E_i(\boldsymbol{k}) = eq(\boldsymbol{x}_i, \boldsymbol{\alpha}_i)$ implies:

$$\big(E_i(\boldsymbol{k}), s_{\sigma,i}(\boldsymbol{k})\big) = \big(eq(\boldsymbol{x}_i, \boldsymbol{\alpha}_i), s_{id}(\boldsymbol{x}_i)\big) \tag{2}$$

A multiplicity counter polynomial $m_i(\boldsymbol{x}_i)$ is then defined to track the frequency of occurrences of $\big(eq(\boldsymbol{x}_i, \boldsymbol{\alpha}_i), s_{id}(\boldsymbol{x}_i)\big)$ across $\boldsymbol{k} \in \{0,1\}^{\log n}$, exactly recording how many times each specific coordinate block $\boldsymbol{x}_i$ appears among the non-zero entries. By Multiset Check Lemma 2, if equation (2) holds for all $\boldsymbol{k} \in \{0,1\}^{\log n}$, then for any randomly chosen challenges $e_{i,1}$ and $e_{i,2}$, we have:

$$\begin{aligned}&\sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} \frac{1}{1 + e_{i,1} \cdot E_i(\boldsymbol{k}) + e_{i,2} \cdot s_{\sigma,i}(\boldsymbol{k})} \\ =\ &\sum_{\boldsymbol{x}_i \in \{0,1\}^{\log m}} \frac{m_i(\boldsymbol{x}_i)}{1 + e_{i,1} \cdot eq(\boldsymbol{x}_i, \boldsymbol{\alpha}_i) + e_{i,2} \cdot s_{id}(\boldsymbol{x}_i)}.\end{aligned} \tag{3}$$

For each $i \in [c]$, we define the functions as follows:

$$\begin{aligned}&f_{2i-1}(\boldsymbol{k}) = 1 + e_{i,1} \cdot E_i(\boldsymbol{k}) + e_{i,2} \cdot s_{\sigma,i}(\boldsymbol{k}), \\ &f_{2i}(\boldsymbol{k}_1, \boldsymbol{k}_2) = 1 + \big(e_{i,1} \cdot eq(\boldsymbol{k}_1, \boldsymbol{\alpha}_i) + e_{i,2} \cdot s_{\mathrm{id}}(\boldsymbol{k}_1)\big) \cdot t(\boldsymbol{k}_2), \\ &m_i'(\boldsymbol{k}_1, \boldsymbol{k}_2) = m_i(\boldsymbol{k}_1) \cdot t(\boldsymbol{k}_2), \quad t(\boldsymbol{k}_2) = \prod_{j \in [\log(n/m)]} k_{2,j},\end{aligned}$$

where $\boldsymbol{k}_1 \in \{0,1\}^{\log m}$ and $\boldsymbol{k}_2 \in \{0,1\}^{\log(n/m)}$. To validate equation (3) for each $i \in [c]$, we reduce the problem to verifying that the following sum equals zero:

$$\sum_{\boldsymbol{k}\in\{0,1\}^{\log n}} \left(\frac{1}{f_{2i-1}(\boldsymbol{k})} - \frac{m_i'(\boldsymbol{k})}{f_{2i}(\boldsymbol{k})} \right) = 0.$$

And then, the verifier selects random challenges $\{e_{i,3} \in \mathbb{F}\}_{i\in[c]}$ and checks:

$$\sum_{\boldsymbol{k}\in\{0,1\}^{\log n}} \sum_{i=1}^{c} e_{i,3} \left(\frac{1}{f_{2i-1}(\boldsymbol{k})} - \frac{m_i'(\boldsymbol{k})}{f_{2i}(\boldsymbol{k})} \right) = 0. \tag{4}$$

By combining the rational function (4) over a common denominator, the problem reduces to verifying that a high-degree polynomial is zero everywhere on the Boolean hypercube. By integrating logarithmic-derivative arguments with DESC PIOP, our framework inherently optimizes this batching process. The DESC PIOP efficiently performs this verification as a data-parallel circuit with a linear-time prover. Crucially, this optimization allows the prover to merge multiple sparse matrix evaluations into a single sum-check instance, significantly reducing commitment overhead.

Definition 10 (SPARK relation). *The relation $\mathcal{R}_{SPARK}$ is defined as the set of tuples*

$$(i; x; \omega) = ((\delta, [[val]]); (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_c, v); val)\,,$$

where the parameter δ specifies the positions of non-zero entries in a sparse matrix M, and val represents $\log n$*-variate degree-1 polynomials that determine the values of these non-zero entries, such that $M(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_c) = v$.*

Construction. The tuple $\big((\delta, [[val]]); (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_c, v); val\big)$ defines a matrix structure where the parameter δ determines the placement of non-zero entries. This configuration generates $2c$ oracles: $[[s_{\sigma,i}]]$ and $[[m_i]]$ for $i \in [c]$. The index polynomials $s_{\sigma,i}(\boldsymbol{k})$ determine the positions of non-zero elements, while the counter polynomials $m_i(\boldsymbol{x})$ track the frequency of these non-zero element positions. The SPARK PIOP protocol is defined as follows:

- The prover $\mathcal{P}$ sends the oracles $\{[[E_i]]\}_{i\in[c]}$ to the verifier $\mathcal{V}$.
- $\mathcal{V}$ sends the challenges $e_{i,1}, e_{i,2}, e_{i,3}$ (for $i \in [c]$) to $\mathcal{P}$.
- $\mathcal{P}$ sends $[[\widetilde{h}]]$ to $\mathcal{V}$, where the multilinear extension $\widetilde{h}(\boldsymbol{k})$ is defined as:

$$\widetilde{h}(\boldsymbol{k}) = \sum_{i\in[c]} e_{i,3} \cdot \left(\frac{1}{f_{2i-1}(\boldsymbol{k})} - \frac{m_i'(\boldsymbol{k})}{f_{2i}(\boldsymbol{k})} \right) \quad \text{for } \boldsymbol{k} \in \{0,1\}^{\log n}, \tag{5}$$

with the polynomials $f_i(\boldsymbol{k})$ and $m_i'(\boldsymbol{k})$ given by:

$$\begin{aligned}
&f_{2i-1}(\boldsymbol{k}) = 1 + e_{i,1} \cdot E_i(\boldsymbol{k}) + e_{i,2} \cdot s_{\sigma,i}(\boldsymbol{k}),\\
&f_{2i}(\boldsymbol{k}_1, \boldsymbol{k}_2) = 1 + \big(e_{i,1} \cdot eq(\boldsymbol{k}_1, \boldsymbol{\alpha}_i) + e_{i,2} \cdot s_{\mathrm{id}}(\boldsymbol{k}_1)\big) \cdot t(\boldsymbol{k}_2),\\
&m_i'(\boldsymbol{k}_1, \boldsymbol{k}_2) = m_i(\boldsymbol{k}_1) \cdot t(\boldsymbol{k}_2), \quad t(\boldsymbol{k}_2) = \prod_{i\in[\log(n/m)]} k_{2,i}.
\end{aligned}$$

- $\mathcal{V}$ sends the challenges $e_4, e_5 \in \mathbb{F}$ and $\boldsymbol{\beta} \in \mathbb{F}^{\log n}$ to $\mathcal{P}$.
- Run a sum-check to check:

$$\sum_{\boldsymbol{k} \in \{0,1\}^{\log n}} val(\boldsymbol{k}) \cdot \widetilde{E}(\boldsymbol{k}) + e_4 \cdot \widetilde{h}(\boldsymbol{k}) + e_5 \cdot \mathrm{eq}(\boldsymbol{\beta}, \boldsymbol{k}) \cdot \left(\widetilde{h}(\boldsymbol{k}) \cdot \widetilde{h}_1(\boldsymbol{k}) + \widetilde{h}_2(\boldsymbol{k}) \right) = 0,$$

  where $\widetilde{E}$, $\widetilde{h}_1$, and $\widetilde{h}_2$ are the multilinear extensions of E, h_1, and h_2:

$$E(\boldsymbol{k}) := \prod_{i=1}^{c} E_i(\boldsymbol{k}), \quad h_1(\boldsymbol{k}) = \prod_{i=1}^{2c} f_i(\boldsymbol{k}), \quad h_2(\boldsymbol{k}) = \sum_{i=1}^{2c} m_i''(\boldsymbol{k}) \cdot \prod_{j \in [2c] \setminus \{i\}} f_j(\boldsymbol{k}),$$

  with

$$\text{for } i \in [2c],\ m_i''(\boldsymbol{k}) = \begin{cases} -e_{(i+1)/2,3} & \text{if } i \text{ is odd,} \\ e_{i/2,3} \cdot m_i'(\boldsymbol{k}) & \text{if } i \text{ is even.} \end{cases}$$

- The $\mathcal{V}$ receives $val(\boldsymbol{\gamma}), \widetilde{E}(\boldsymbol{\gamma}), \widetilde{h}(\boldsymbol{\gamma}), \widetilde{h}_1(\boldsymbol{\gamma}), \widetilde{h}_2(\boldsymbol{\gamma})$ to verify the consistency of the messages in the final round of the sum-check protocol, where $\boldsymbol{\gamma} \in \mathbb{F}^{\log n}$ is a randomly sampled challenge. The $\mathcal{V}$ then queries oracles $[[val]]$ and $[[\widetilde{h}]]$ to confirm the correctness of $val(\boldsymbol{\gamma})$ and $\widetilde{h}(\boldsymbol{\gamma})$.
- Run 3 GKR in parallel for:

$$\left(E; \left(\left[\left[\widetilde{V}_{E,\mathrm{in}}\right]\right], \widetilde{E}(\boldsymbol{\gamma})\right); \widetilde{V}_{E,\mathrm{in}}\right), \quad \left(h_1; \left(\left[\left[\widetilde{V}_{h_1,\mathrm{in}}\right]\right], \widetilde{h}_1(\boldsymbol{\gamma})\right); \widetilde{V}_{h_1,\mathrm{in}}\right),$$
$$\left(h_2; \left(\left[\left[\widetilde{V}_{h_2,\mathrm{in}}\right]\right], \widetilde{h}_2(\boldsymbol{\gamma})\right); \widetilde{V}_{h_2,\mathrm{in}}\right) \in \mathcal{R}_{\mathrm{GKR}}.$$

  As shown in Subsection 5.2, the layered arithmetic subcircuits for h_1 and h_2 have size $O(c)$. We have also defined the input functions for the circuit composed of these subcircuits in parallel. The subcircuit E, which shares a similar structure to h_1, also has a size of $O(c)$.
- The $\mathcal{V}$ queries oracles $\{[[s_{\sigma,i}]], [[m_i]], [[E_i]]\}_{i \in [c]}$ to validate the correctness of the input layer's claim.

Theorem 6. *The PIOP for $\mathcal{R}_{SPARK}$ has completeness error $\delta_c = O\left(c \cdot \log n / |\mathbb{F}|\right)$ and has knowledge error $\delta_{spark}^{c,n} := O\left(c + \log^2 c + \log c \cdot \log n) / |\mathbb{F}|\right)$.*

Proof. **Completeness error:** The equation (5) exhibits a completeness error of $O\left(c \cdot \log n / |\mathbb{F}|\right)$, since the probability that $f_i = 0$ is at most $\log n / |\mathbb{F}|$ for each $i \in [2c]$. Notably, the PIOPs for sum-check exhibit perfect completeness. Consequently, the completeness error is upper bounded by $O\left(c \cdot \log n / |\mathbb{F}|\right)$.

Knowledge error: The batch verification process, as demonstrated in equation (5), has a knowledge error of c. Specifically, the PIOP for $\mathcal{R}_{\mathrm{SUM}}$ has a knowledge error of $O\left(\log n / |\mathbb{F}|\right)$. Similarly, the PIOP for $\mathcal{R}_{\mathrm{GKR}}$ demonstrates a soundness error of $O\left(\log(c) \cdot \log(c \cdot n) / |\mathbb{F}|\right)$. Consequently, the overall soundness error is bounded by $O\left((c + \log^2 c + \log c \cdot \log n) / |\mathbb{F}|\right)$. □

Complexity. The layered arithmetic subcircuits for E, h_1, and h_2 are all of size $O(c)$. When we include the cost of the GKR protocol, the overall complexity of the SPARK PIOP for $\mathcal{R}_{\mathrm{SPARK}}$ is as follows:

- The prover time is $\mathrm{tp}_{\mathrm{spark}} = O(cn)$.
- The verifier time is $\mathrm{tv}_{\mathrm{spark}} = O(c + \log^2 c + \log c \cdot \log n)$.
- The total proof size is $\mathrm{r}_{\mathrm{spark}} = O(c + \log^2 c + \log c \cdot \log n)$.
- The query complexity and the number of proof oracles is $3c + 2$, with the oracles being $\{[[m_i]],\ [[s_{\sigma,i}]],\ [[E_i]]\}_{i\in[c]},\ [[val]]$ and $[[\widetilde{h}]]$.

SPARK. SPARK PIOP is then instantiated using a polynomial commitment scheme to produce a sparse polynomial commitment, which requires commitments to $2c + 2$ dense multilinear polynomials over $\log n$ variables and c dense polynomials over $\log m$ variables.

Theorem 7. *Assuming the existence of a polynomial commitment scheme for $(\log m)$-variate multilinear polynomials characterized by the following parameters (with m being a positive integer and, without loss of generality, a power of 2):*

- *the size of the polynomial commitment is* $\mathsf{pc}(m)$*;*
- *the time of the commit algorithm is* $\mathsf{tc}(m)$*;*
- *the time of the prover to prove a polynomial evaluation is* $\mathsf{tp}(m)$*;*
- *the time of the verifier to verify a polynomial evaluation is* $\mathsf{tv}(m)$*;*
- *the proof size is* $\mathsf{p}(m)$*,*

there exists a polynomial commitment scheme for multilinear polynomials over $c \log m$ variables that evaluate to a non-zero value at at most n locations over the Boolean hypercube $\{0,1\}^{c\log m}$, with the following parameters:

- *the size of the sparse polynomial commitment is* $(2c+2)\cdot\mathsf{pc}(n) + c\cdot\mathsf{pc}(m)$*;*
- *the time of the commit algorithm is* $O\left(c\cdot(\mathsf{tc}(n) + \mathsf{tc}(m))\right)$*;*
- *the prover time to prove a polynomial evaluation is* $O(c\cdot(\mathsf{tp}(n) + \mathsf{tp}(m)))$*;*
- *the verifier time to verify a polynomial evaluation is* $O(c\cdot(\mathsf{tv}(n) + \mathsf{tv}(m)))$*;*
- *the proof size is* $O(c\cdot(\mathsf{p}(n) + \mathsf{p}(m)))$*.*

5.2 Constructing Layered Circuits for h_1 and h_2

Without loss of generality, we assume c is a power of 2. If not, we extend the definitions of $f_{2i-1}(\boldsymbol{k})$, $f_{2i}(\boldsymbol{k})$, and $m'_i(\boldsymbol{k})$ for $i \in \{c+1, \ldots, 2^{\lceil \log c \rceil}\}$ as follows:

$$f_{2i-1}(\boldsymbol{k}) = f_{2i}(\boldsymbol{k}) = m'_i(\boldsymbol{k}) = 1 \quad \text{for all } \boldsymbol{k} \in \{0,1\}^{\log n}.$$

Finally, Equation (4) can be transformed into the following equation:

$$\sum_{\boldsymbol{k}\in\{0,1\}^{\log n}} \sum_{i\in[2^{\lceil \log c\rceil}]} e_{i,3}\cdot\left(\frac{1}{f_{2i-1}(\boldsymbol{k})} - \frac{m'_i(\boldsymbol{k})}{f_{2i}(\boldsymbol{k})}\right) = 0.$$

To verify this relation, we define a multilinear polynomial $\widetilde{h}(\boldsymbol{k})$ and reduce the problem to verifying the following two conditions:

$$\sum_{\boldsymbol{k}\in\{0,1\}^{\log n}} \widetilde{h}(\boldsymbol{k}) = 0,$$

and

$$\widetilde{h}(\boldsymbol{k}) = \sum_{i\in[c]} e_{i,3} \cdot \left(\frac{1}{f_{2i-1}(\boldsymbol{k})} - \frac{m_i'(\boldsymbol{k})}{f_{2i}(\boldsymbol{k})} \right) \quad \text{for all } \boldsymbol{k} \in \{0,1\}^{\log n}.$$

We can directly apply the sum-check protocol to verify the first condition. For the second condition, we express it as a high-degree polynomial identity:

$$\widetilde{h}(\boldsymbol{k}) \cdot h_1(\boldsymbol{k}) + h_2(\boldsymbol{k}) = 0 \quad \text{for all } \boldsymbol{k} \in \{0,1\}^{\log n},$$

where

$$\begin{aligned} h_1(\boldsymbol{k}) &:= g_1\big(f_1(\boldsymbol{k}), \ldots, f_{2c}(\boldsymbol{k})\big) = \prod_{i=1}^{2c} f_i(\boldsymbol{k}), \\ h_2(\boldsymbol{k}) &:= g_2\big(m_1''(\boldsymbol{k}), \ldots, m_{2c}''(\boldsymbol{k}), f_1(\boldsymbol{k}), \ldots, f_{2c}(\boldsymbol{k})\big) \\ &= \sum_{i=1}^{2c} m_i''(\boldsymbol{k}) \cdot \prod_{j\in[2c]\setminus\{i\}} f_j(\boldsymbol{k}), \end{aligned}$$

with

$$m_i''(\boldsymbol{k}) = \begin{cases} -e_{(i+1)/2,3} & \text{when } i \text{ is odd}, \\ e_{i/2,3} \cdot m_i'(\boldsymbol{k}) & \text{when } i \text{ is even}. \end{cases}$$

Subsequently, a random challenge $\boldsymbol{\alpha} \in \mathbb{F}^{\log n}$ is selected, and a sum-check protocol is employed to confirm the relation:

$$\sum_{\boldsymbol{k}\in\{0,1\}^{\log n}} eq(\boldsymbol{\alpha}, \boldsymbol{k})\big(\widetilde{h}(\boldsymbol{k}) \cdot h_1(\boldsymbol{k}) + h_2(\boldsymbol{k})\big) = 0.$$

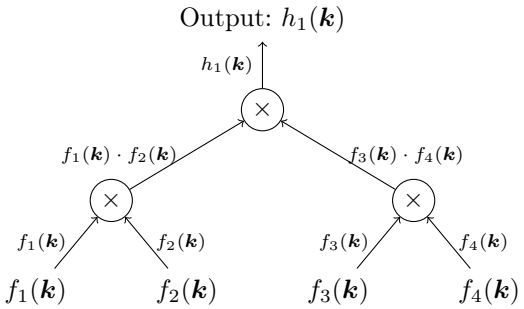
To validate the sum to 0, we can employ the DESC PIOP. Since h is a multilinear polynomial, we transform the computations of h_1 and h_2 into two arithmetic subcircuits C_1 and C_2, respectively. The verification reduces to checking whether the following relation holds:

$$\sum_{\boldsymbol{k}\in\{0,1\}^{\log n}} eq(\boldsymbol{\alpha}, \boldsymbol{k})\big(\widetilde{h}(\boldsymbol{k}) \cdot \widetilde{h}_1(\boldsymbol{k}) + \widetilde{h}_2(\boldsymbol{k})\big) = 0.$$

Here, $\widetilde{h}_1(\boldsymbol{k})$ and $\widetilde{h}_2(\boldsymbol{k})$ represent the multilinear extensions of the outputs from C_1 and C_2, respectively. Each of these circuits consists of n subcircuits h_1 and h_2 operating in parallel. The multilinear extensions of the inputs are defined as:

$$\begin{aligned} \widetilde{V}_{1,\log(2c)}(\boldsymbol{x}, \boldsymbol{k}) &= \sum_{i\in[2c]} eq(\boldsymbol{x}, \langle i\rangle) \cdot f_i(\boldsymbol{k}), \\ \widetilde{V}_{2,\log(2c)}(\boldsymbol{x}, \boldsymbol{k}) &= \sum_{i\in[2c]} eq(\boldsymbol{x}, \langle i\rangle) \cdot m_i''(\boldsymbol{k}), \end{aligned}$$

where $\boldsymbol{x} \in \{0,1\}^{\log(2c)}$, $\boldsymbol{k} \in \{0,1\}^{\log n}$, and $\langle i\rangle$ denotes the binary representation of i. To illustrate the construction of C_1 and C_2, consider a simple case where $c = 2$. The computation of subcircuit h_1 is defined layer by layer as follows:



Subsequently, we can infer that for layer i ($i \in [2]$) in circuit C_1, the following holds for $\boldsymbol{z} \in \{0,1\}^{i-1+\log n}$:

$$\widetilde{V}_{1,i-1}(\boldsymbol{z}) = \widetilde{V}_{1,i}(0,\boldsymbol{z}) \cdot \widetilde{V}_{1,i}(1,\boldsymbol{z}). \tag{6}$$

The computation of subcircuit h_2 is defined based on the computation trajectory of h_1. Specifically, we have:

$$\sum_{i=1}^{4} m_i''(\boldsymbol{k}) \cdot \prod_{j \in [4]\setminus\{i\}} f_j(\boldsymbol{k}) = (m_1''(\boldsymbol{k}) \cdot f_2(\boldsymbol{k}) + m_2''(\boldsymbol{k}) \cdot f_1(\boldsymbol{k})) \cdot f_3(\boldsymbol{k}) \cdot f_4(\boldsymbol{k}) + (m_3''(\boldsymbol{k}) \cdot f_4(\boldsymbol{k}) + m_4''(\boldsymbol{k}) \cdot f_3(\boldsymbol{k})) \cdot f_1(\boldsymbol{k}) \cdot f_2(\boldsymbol{k}).$$

From this, we generalize the output at layer i (where $i \in [2]$) of C_2 as:

$$\widetilde{V}_{2,i-1}(\boldsymbol{x}) = \widetilde{V}_{2,i}(0,\boldsymbol{x}) \cdot \widetilde{V}_{1,i}(1,\boldsymbol{x}) + \widetilde{V}_{2,i}(1,\boldsymbol{x}) \cdot \widetilde{V}_{1,i}(0,\boldsymbol{x}). \tag{7}$$

For any $c > 2$, we can similarly deduce from Equations (6), (7) that the circuits C_1 and C_2 satisfy:

$$\widetilde{V}_{1,i-1}(\boldsymbol{x}) = \widetilde{V}_{1,i}(0,\boldsymbol{x}) \cdot \widetilde{V}_{1,i}(1,\boldsymbol{x}),$$

$$\widetilde{V}_{2,i-1}(\boldsymbol{x}) = \widetilde{V}_{2,i}(0,\boldsymbol{x}) \cdot \widetilde{V}_{1,i}(1,\boldsymbol{x}) + \widetilde{V}_{2,i}(1,\boldsymbol{x}) \cdot \widetilde{V}_{1,i}(0,\boldsymbol{x}).$$

From this structure, it follows that each layer i in C_1 and C_2 has a width of $2^i \cdot n$; and the subcircuits $\widetilde{h}_1$ and $\widetilde{h}_2$ have a size of $O(c)$, with the total depth being $\log(2c)$.

6 Doubly Efficient Interactive Proofs

In the previous sections, we demonstrated that the quasi-linear prover overhead in high-degree Sum-Checks is not inherent, but rather a byproduct of how polynomial constraints are represented. By reorganizing these constraints into data-parallel circuits, DESC PIOP achieved linear prover time. We explicitly define DESC as an optimized method for efficiently proving evaluations of high-degree virtual polynomials [11].

This success prompts a broader question: Can this "constraint-first" perspective optimize other aspects of GKR-based proofs? Here, we revisit the GKR protocol. Rather than claiming a unifying GKR framework, we focus on generalizing widely used structures—such as product circuits ($Y = \prod_{i=1}^{n} X_i$)—into a communication-efficient instantiation for arithmetic circuits. By incorporating CCS-style algebraic structure, we express each layer using matrix–vector multiplications, Hadamard products, and summations.

The key insight is that GKR efficiency depends on the algebraic organization of constraints, not on a strict nesting of sum-checks. Restructuring constraints in this way improves aggregation without changing the underlying proof paradigm, preserving linear prover time while reducing communication overhead.

6.1 An Improved GKR Protocol for Layered Circuits

The original GKR protocol verifies a depth-d layered arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ by recursively applying sum-checks to $\widetilde{V}_i$, the multilinear extension of the values at layer i.

Whereas the standard GKR layer reduction takes the form $\widetilde{V}_i(\boldsymbol{z}) = \sum_{\boldsymbol{x}} \mathrm{eq}(\boldsymbol{z}, \boldsymbol{x}) \cdot \widetilde{V}_{i+1}(0, \boldsymbol{x}) \cdot \widetilde{V}_{i+1}(1, \boldsymbol{x})$, we generalize this relation to efficiently capture structured constraints:

$$\widetilde{V}_i(\boldsymbol{z}) = \sum_{\boldsymbol{x} \in \{0,1\}^{s_i}} \mathrm{eq}(\boldsymbol{z}, \boldsymbol{x}) \cdot f_i(\boldsymbol{x}),$$

where $f_i(\boldsymbol{x})$ is defined as the Hadamard product of matrix-vector multiplications over layer $i+1$:

$$f_i(\boldsymbol{x}) = \left(\sum_{\boldsymbol{y} \in \{0,1\}^{s_{i+1}}} \widetilde{A}_{i+1}(\boldsymbol{x}, \boldsymbol{y}) \cdot \widetilde{V}_{i+1}(\boldsymbol{y}) \right) \cdot \left(\sum_{\boldsymbol{y} \in \{0,1\}^{s_{i+1}}} \widetilde{B}_{i+1}(\boldsymbol{x}, \boldsymbol{y}) \cdot \widetilde{V}_{i+1}(\boldsymbol{y}) \right).$$

Since the evaluations of $\sum \widetilde{A}\widetilde{V}_{i+1}$ and $\sum \widetilde{B}\widetilde{V}_{i+1}$ at $\boldsymbol{x}$ naturally substitute for $\widetilde{V}_{i+1}(0, \boldsymbol{x})$ and $\widetilde{V}_{i+1}(1, \boldsymbol{x})$, we can reduce them into a single evaluation claim for $\widetilde{V}_{i+1}$ via a random linear combination. This elegantly bypasses the need for an additional Sum-check protocol over $\boldsymbol{y}$, providing a framework that sufficiently covers various constraint representations. This reformulation is sound, since the R1CS naturally captures both additive and multiplicative relations between linear combinations of variables [28].

To optimize proof size while strictly guaranteeing a prover time linear in field operations, we further introduce CCS to support higher-degree compositions. For example, extending a layer reduction modeled as the product of multiple evaluation points (e.g., the product of four points $\widetilde{V}_{i+1}(0, 0, \boldsymbol{x})$, $\widetilde{V}_{i+1}(1, 0, \boldsymbol{x})$, $\widetilde{V}_{i+1}(0, 1, \boldsymbol{x})$, $\widetilde{V}_{i+1}(1, 1, \boldsymbol{x})$) allows $\widetilde{V}_i$ to be expressed as a degree-c composition:

$$\widetilde{V}_i(\boldsymbol{z}) = \sum_{\boldsymbol{x} \in \{0,1\}^{s_i}} \mathrm{eq}(\boldsymbol{z}, \boldsymbol{x}) \cdot \prod_{j=1}^{c} \left(\sum_{\boldsymbol{y} \in \{0,1\}^{s_{i+1}}} M_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \widetilde{V}_{i+1}(\boldsymbol{y}) \right),$$

where $M_1, \ldots, M_c \in \mathbb{F}^{m \times m}$ are sparse matrices. When $c = 4$, this higher-degree grouping reduces the number of Sum-check invocations to $\log_4 n$. This structure naturally extends to the full CCS framework to support weighted sums of such products, decreasing communication complexity and providing a highly flexible prover-time/communication tradeoff.

The protocol implements the layer-to-layer reduction via two sum-check invocations. First, the claim $\widetilde{V}_i(\boldsymbol{u}_i) = \sum_{\boldsymbol{x}} \mathsf{eq}(\boldsymbol{u}_i, \boldsymbol{x}) \cdot f_i(\boldsymbol{x})$ is reduced to a single evaluation point $\boldsymbol{v}_i \in \mathbb{F}^{s_i}$, yielding $m_j(\boldsymbol{v}_i) = \sum_{\boldsymbol{y}_{i+1}} \widetilde{M_j}(\boldsymbol{v}_i, \boldsymbol{y}_{i+1}) \cdot \widetilde{V}_{i+1}(\boldsymbol{y}_{i+1})$ for $j \in [c]$.

Next, a random linear combination $\boldsymbol{e} \in \mathbb{F}^c$ aggregates these c equations into a single claim:

$$\sum_{j=1}^{c} e_j \cdot m_j(\boldsymbol{v}_i) = \sum_{\boldsymbol{y}_{i+1} \in \{0,1\}^{s_{i+1}}} \left(\sum_{j=1}^{c} e_j \cdot \widetilde{M_j}(\boldsymbol{v}_i, \boldsymbol{y}_{i+1}) \right) \widetilde{V}_{i+1}(\boldsymbol{y}_{i+1}).$$

A second sum-check reduces this aggregated sum to a single evaluation $\widetilde{V}_{i+1}(\boldsymbol{u}_{i+1})$ at a random point $\boldsymbol{u}_{i+1}$. This completes the transition to layer $i + 1$, operating on structured algebraic constraints rather than individual gates.

Comparison with the Original GKR Protocol. In the original GKR protocol, a depth-d layered arithmetic circuit of width S is verified gate by gate. At each layer, sum-check is applied to a multivariate polynomial in $2 \log S$ variables, corresponding to the binary encodings of the two input wires of each gate. Each variable appears with degree at most 2. Consequently, each round of the sum-check protocol communicates 3 field elements, resulting in a communication cost of $6 \log S$ per layer and $6d \log S$ overall, with prover time $O(dS)$. For uniform circuits, the verifier can evaluate the wiring predicates at each layer in time $O(\log S)$, yielding an overall verifier time of $O(d \log S)$.

Our formulation preserves the GKR proof paradigm but reorganizes how arithmetic constraints are encoded. Instead of explicit addition and multiplication gates, each layer is expressed using summations, sparse matrix–vector multiplications, and Hadamard products. Importantly, addition constraints are absorbed into linear combinations and do not contribute to circuit depth; only multiplicative dependencies induce new sum-check layers.

For example, consider a binary-tree summation computing $y = \sum_{i=1}^{n} x_i$ with $n = 2^k$. Under the standard GKR formulation, this computation is realized by a depth-$\log n$ circuit of addition gates, incurring $\log n$ layers of sum-check. In contrast, under an R1CS or CCS-style encoding, the same computation is represented as a single linear constraint, resulting in constant effective depth.

Similarly, consider a multiplication tree computing $y = \prod_{i=1}^{n} x_i$ with $n = 2^k$. In the standard GKR protocol, this requires a binary multiplication tree of depth $d = \log n$, leading to $\log n$ sum-check invocations. In our formulation, these multiplications are aggregated into bounded-degree algebraic constraints. By grouping at most d_1 multiplicative terms, the effective depth is reduced to $\log_{d_1}(n)$. Consequently, the first sum-check at each effective layer involves a polynomial of degree $d_1 + 1$, while subsequent aggregated claims have degree 2.

The resulting total communication complexity is

$$(d_1 + 5)\,\log S \cdot \log_{d_1}(2^d),$$

which is minimized at $d_1 = 6$, yielding approximately $4.25\,d \log S$.

This improvement arises from a re-encoding of the circuit rather than the introduction of a new proof system. In particular, for uniform circuits, the sparse matrix predicates admit efficient evaluation by the verifier, analogous to the evaluation of the $\widetilde{add}$ and $\widetilde{mult}$ wiring predicates in the original GKR protocol. As a result, the protocol preserves linear prover time and efficient verification.

Comparison with Optimized Sum-check Techniques. Since Sum-check and GKR serve as sub-protocols, existing state-of-the-art optimizations integrate seamlessly into our framework. These optimizations include utilizing binary-tree topologies, sending d evaluations per round, or alternatively, leveraging polynomial commitments (e.g., sending commitments to degree-d univariate polynomials for final batch verification). By combining these techniques with our CCS-based GKR, DESC achieves better proof size than standard Sum-check under identical optimizations, while consistently offering flexible trade-offs between communication and prover time.

6.2 DE-PIOP for General Arithmetic Circuits

In this section, we present a generalized DEIP protocol specifically designed for general arithmetic circuits, referred to as DE-PIOP. The protocol is designed to validate a claimed value v as the output of a general arithmetic circuit.

As demonstrated in [31], in a general arithmetic circuit, each gate within layer i can receive outputs from gates spanning layers $i+1$ through d. To accommodate this layered dependency, a key assumption is that every gate at layer i must receive at least one input from layer $i+1$, designated as the left operand. Under this assumption, the multilinear extension of the output $\widetilde{V}_i$ in layer $(i+1)$ can be re-expressed as:

$$\begin{aligned}\widetilde{V}_i(\boldsymbol{z}) = \sum_{j=i+1}^{d} \sum_{\substack{\boldsymbol{y}_j \in \{0,1\}^{s_j} \\ \boldsymbol{x}_{i+1} \in \{0,1\}^{s_{i+1}}}} & \widetilde{add}_{i+1,j}(\boldsymbol{z}_i, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j)\big(\widetilde{V}_{i+1}(\boldsymbol{x}_{i+1}) + \widetilde{V}_j(\boldsymbol{y}_j)\big) \\ & + \widetilde{mult}_{i+1,j}(\boldsymbol{z}_i, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j)\big(\widetilde{V}_{i+1}(\boldsymbol{x}_{i+1})\widetilde{V}_j(\boldsymbol{y}_j)\big),\end{aligned}$$

where $\widetilde{add}_{i+1,j}(\boldsymbol{z}_i, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j) = 1$ ($\widetilde{mult}_{i+1,j}(\boldsymbol{z}_i, \boldsymbol{x}_{i+1}, \boldsymbol{y}_j) = 1$) if and only if gate $\boldsymbol{z}_i$ in layer $i+1$ is the addition (multiplication) of values $\widetilde{V}_{i+1}(\boldsymbol{x}_{i+1})$ and $\widetilde{V}_j(\boldsymbol{y}_j)$.

We generalize this formulation by assuming that the inputs to each gate in layer i are linear combinations of outputs from layers $i+1$ through d. Under this assumption, we redefine $V_i(\boldsymbol{z})$ as follows:

$$\widetilde{V}_i(\boldsymbol{z}) := \sum_{\boldsymbol{x}_i \in \{0,1\}^{s_i}} f_i(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_i \in \{0,1\}^{s_i}} eq(\boldsymbol{z}, \boldsymbol{x}_i) \cdot f_{i,a}(\boldsymbol{x}_i) \cdot f_{i,b}(\boldsymbol{x}_i),$$

where

$$f_{i,a}(\boldsymbol{x}_i) = \sum_{j=i+1}^{d} \sum_{\boldsymbol{y}_j \in \{0,1\}^{s_j}} \widetilde{A}_{i+1,j}(\boldsymbol{x}_i, \boldsymbol{y}_j)\widetilde{V}_j(\boldsymbol{y}_j),$$

$$f_{i,b}(\boldsymbol{x}_i) = \sum_{j=i+1}^{d} \sum_{\boldsymbol{y}_j \in \{0,1\}^{s_j}} \widetilde{B}_{i+1,j}(\boldsymbol{x}_i, \boldsymbol{y}_j)\widetilde{V}_j(\boldsymbol{y}_j).$$

By following the methodology of [31], it is straightforward to derive a corresponding doubly-efficient interactive protocol for this generalized construction.

7 Conclusion

Our work presents an efficient Sum-Check framework built upon the GKR protocol: efficiency is fundamentally determined by the algebraic organization of constraints. We validated this principle in two directions: first, through DESC PIOP (Section 3), which eliminates the prover bottleneck for high-degree polynomials by treating them as structured circuits; and second, through a refined GKR protocol (Section 6), which reduces communication overhead by aggregating circuit layers. Together, these results suggest that future optimizations should focus on the algebraic encoding of the computation rather than the proof protocol itself.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work is supported by the National Natural Science Foundation of China (Grant No. 62272269) and the Scientific Research Innovation Capability Support Project for Young Faculty (Grant No. ZYGXQNJSKYCXNLZCXM-I21).

References

1. Baldimtsi, F., Chalkias, K.K., Ji, Y., Lindstrøm, J., Maram, D., Riva, B., Roy, A., Sedaghat, M., Wang, J.: zklogin: Privacy-preserving blockchain authentication with existing credentials. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 3182–3196 (2024)
2. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Annual international cryptology conference. pp. 701–732. Springer (2019)
3. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: 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. pp. 103–128. Springer (2019)
4. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14. pp. 31–60. Springer (2016)

5. Bootle, J., Chiesa, A., Hu, Y., Orru, M.: Gemini: Elastic snarks for diverse environments. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 427–457. Springer (2022)
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
7. Buterin, V.: An incomplete guide to rollups. Vitalik Buterin website (2021)
8. Campanelli, M., Faonio, A., Fiore, D., Li, T., Lipmaa, H.: Lookup arguments: improvements, extensions and applications to zero-knowledge decision trees. In: IACR International Conference on Public-Key Cryptography. pp. 337–369. Springer (2024)
9. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 499–530. Springer (2023)
10. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. 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. pp. 769–793. Springer (2020)
11. Diamond, B.E., Posen, J.: Succinct arguments over towers of binary fields. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 93–122. Springer (2025)
12. Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. Cryptology ePrint Archive (2022)
13. Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive (2020)
14. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. p. 953 (2019)
15. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. Journal of the ACM (JACM) **62**(4), 1–64 (2015)
16. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for r1cs. In: Annual International Cryptology Conference. pp. 193–226. Springer (2023)
17. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
18. Haböck, U.: Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive (2022)
19. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM (JACM) **39**(4), 859–868 (1992)
20. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2111–2128 (2019)
21. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE symposium on security and privacy. pp. 459–474. IEEE (2014)
22. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **27**(4), 701–717 (1980)

23. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)
24. Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive (2023)
25. Setty, S., Thaler, J., Wahby, R.: Unlocking the lookup singularity with lasso. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 180–209. Springer (2024)
26. Shoup, V.: Lower bounds for discrete logatithms and related problems. In: EUROCRYPT 1997. pp. 256–266 (1997)
27. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Annual Cryptology Conference. pp. 71–89. Springer (2013)
28. Thaler, J.: Proofs, arguments, and zero-knowledge. Foundations and Trends® in Privacy and Security **4**(2–4), 117–660 (2022)
29. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: 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. pp. 733–764. Springer (2019)
30. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 3003–3017 (2022)
31. Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., Zhang, Y.: Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 159–177 (2021)