PANCAKE: A SNARK with Plonkish Constraints, Almost-Free Additions, No Permutation Check, and a Linear-Time Prover

Yuxi Xue[1], Peimin Gao[1], Xingye Lu[1], and Man Ho Au[1]

Department of Computing, The Hong Kong Polytechnic University, Hong Kong
yuxi-ivy.xue@connect.polyu.hk, peimin-cs.gao@connect.polyu.hk,
xing-ye.lu@polyu.edu.hk, mhaau@polyu.edu.hk

Abstract. We present Pancake, a linear-time SNARK with a circuit-specific setup that eliminates the explicit representation and separate verification of addition gates in Plonkish constraint systems. Specifically, we consolidate wiring constraints and addition-gate constraints into a single family of general linear constraints, which can be enforced efficiently via a single sumcheck protocol. As a result, Pancake achieves "almost-free" addition gates, which significantly reduces the witness size and directly improves prover efficiency while preserving full support for high-degree custom gates.
Our implementation shows that Pancake outperforms the state-of-the-art Plonkish SNARK HyperPlonk (Chen et al., EUROCRYPT 2023) in terms of prover efficiency. For a circuit size of 2^{24} where half the gates are additions, Pancake achieves prover speedups of 1.67× (single-threaded) and 2.43× (32-threaded), while also generating smaller proofs and maintaining comparable verification time.

Keywords: SNARK · Linear-time prover · Plonkish constraint systems · Polynomial IOP

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) [30,18,5] are cryptographic primitives that enable the prover to generate a proof to convince the verifier of the validity of a computation. Typically, they are used to prove statements of the form: "Given a function F and a public input x, there exists a private witness w such that $F(x, w) = 1$." A key feature of SNARKs is their succinctness: the proof size and verification cost are constant or sublinear in the length of the witness. When combined with the zero-knowledge (ZK) property [19,4], SNARKs allow a prover to demonstrate the validity of a computation without leaking any additional information. Due to these powerful features, SNARKs have enabled a wide range of applications, including scalable blockchain systems [51,35], private smart contracts [38,28], and verifiable outsourced computation [22].

Modern SNARK constructions typically begin by encoding the computational statement as an arithmetic circuit defined over a finite field. This circuit

is then formulated as a constraint system comprising a collection of algebraic relations, whose satisfiability guarantees the correct execution of the original computation. Two widely used constraint systems are Rank-1 Constraint Systems (R1CS) [17,40,2] and Plonkish constraint systems [15,11,13], each with distinct trade-offs.

R1CS consists of a set of quadratic constraints, where each constraint verifies a single multiplication between two linear combinations of variables. This structure allows linear operations, such as addition, to be effectively "free," as they are implicitly absorbed into the combinations without increasing the number of constraints. However, R1CS is limited to degree-2 relations, meaning that higher-degree relations must be decomposed into multiple intermediate constraints.

In contrast, Plonkish constraint systems adopt a gate-centric architecture, defining constraints in two categories: (1) *gate constraints* that enforce the local correctness of each gate, and (2) *wiring constraints* that ensure the output of one gate is correctly propagated as the input to subsequent gates. This design natively supports high-degree custom gates, significantly enhancing expressiveness. Specifically, a single custom gate can encapsulate complex operations, such as a round of a cryptographic hash (e.g., Poseidon hash [21]) or an elliptic curve group operation [24], thereby reducing the constraint count for complex operations.

Nevertheless, unlike R1CS, addition operations in Plonkish are not free, as each consumes an explicit constraint. This overhead is substantial in practice; for widely used circuits such as Poseidon hash [21], SHA-256 [27], and AES-128 [37], addition gates account for 50-70% of the total gate count. Since the prover cost is proportional to the number of constraints, these explicit addition constraints represent a large fraction of the total proving cost in Plonkish systems.

While the recent work Garuda [14] demonstrates that linear-time proving with inexpensive linear computation and high-degree custom gates is achievable in the Generalized R1CS (GR1CS) system, a central challenge remains for Plonkish SNARKs: can one attain comparable prover efficiency while retaining its native circuit model and toolchain? This question is practically significant because a wide range of real-world applications are engineered around Plonkish-style custom gates and circuit descriptions. In these existing deployments, wiring consistency and abundant linear computation are still realized and checked through mechanisms that introduce substantial prover overhead.

These considerations motivate the development of a Plonkish-compatible SNARK that retains native support for high-degree custom gates, makes linear computation essentially free, and enables linear-time proof generation.

In this paper, we introduce Pancake (**P**lonkish, **A**lmost-free addition gate, **N**o permutation **C**heck, succinct non-interactive **A**rgument of **K**nowledg**E**), a linear-time SNARK with a circuit-specific setup that directly addresses this challenge. Our core idea is to absorb linear computation into the wiring relations themselves. Specifically, we directly constrain the inputs to non-linear gates to satisfy the required linear relations (e.g., enforcing that a multiplication input equals the sum of earlier values), rather than introducing dedicated addition gates and wiring them forward. Consequently, the constraint system retains only

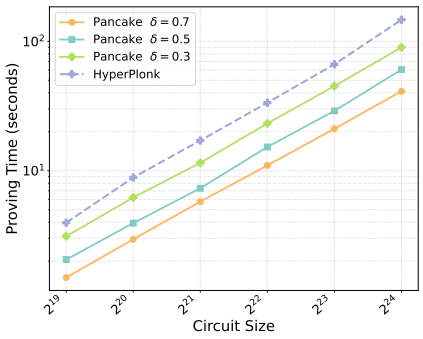
non-linear gate constraints, while additions are handled implicitly through linear consistency checks, thereby making additions essentially free without sacrificing Plonkish-style expressiveness for custom gates. The novelty is therefore at the arithmetization level: rather than optimizing HyperPlonk's permutation argument, Pancake removes the need for it by representing wiring consistency and addition semantics as a unified family of linear constraints.

To achieve a linear-time prover, Pancake operates within the multivariate polynomial framework, encoding the circuit's execution trace as multilinear polynomials over the Boolean hypercube. By operating over this domain rather than using univariate encodings, Pancake avoids the quasi-linear overhead of Fast Fourier Transforms (FFTs) typically required in univariate settings, thereby removing a major bottleneck to achieving linear-time proof generation.

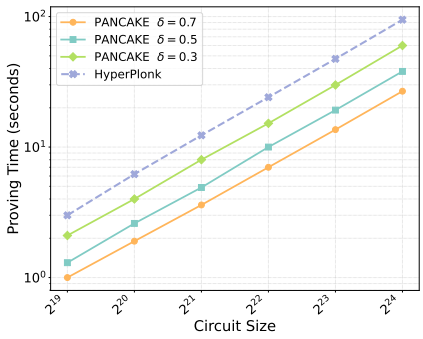
The integration of this efficient encoding with our new constraint system yields Pancake: a SNARK that preserves Plonkish expressiveness for high-degree custom gates while delivering a linear-time prover and an "almost-free" treatment of linear operations.

We implement Pancake and benchmark it against HyperPlonk [11], the state-of-the-art multilinear SNARK based on Plonkish constraints. Our evaluation covers both vanilla Plonk circuit and Jellyfish Turbo Plonk circuit (the latter supporting high-degree custom gates), varying the addition gate ratio δ. As shown in Fig. 1, Pancake consistently outperforms HyperPlonk in all tested configurations. In single-threaded settings for Jellyfish circuits ($n = 2^{24}$), Pancake achieves speedups of 2.37× and 1.67× for $\delta = 0.7$ and $\delta = 0.5$, respectively. Even when the proportion of addition gates is low ($\delta = 0.3$), Pancake maintains a performance advantage with a 1.11× speedup. This advantage is further amplified in parallel settings: with 32 threads, the speedup at $\delta = 0.5$ increases to 2.43×, demonstrating Pancake's superior scalability in parallelized environments. A limitation of Pancake is its requirement for a circuit-specific setup. While universal setups offer broader applicability, circuit-specific setups are a practical and well-accepted choice for long-lived applications like zkRollups, privacy-preserving Layer 2 solutions, and virtual machines, where the prover's cost is the dominant concern. In such large-scale scenarios, Pancake's fast proving represents a meaningful and justified trade-off.

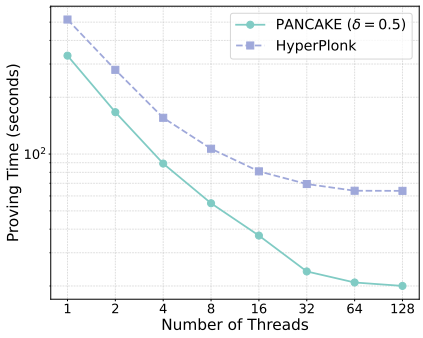
To situate Pancake among circuit-specific SNARKs, we include two additional systems representing distinct design paradigms. First, we include Groth16 [23] as a canonical baseline, which remains the industry standard for proof size and verification efficiency. Second, we consider Garuda [14], a recent GR1CS-based SNARK that achieves linear-time proving. However, given that direct comparisons between differing constraint systems (Plonkish vs. R1CS/GR1CS) are inherently complex and sensitive to implementation details, we defer a detailed case study on the Rescue hash circuit to Appendix B. There, Garuda achieves the fastest proving time, followed closely by Pancake, with both substantially outperforming Groth16.



(a) Jellyfish Turbo (32 threads)



(b) Vanilla Plonk (32 threads)



(c) Jellyfish Turbo (2^{22} circuit size)

Fig. 1. Proving times for HyperPlonk and Pancake across different addition gate ratios.

1.1 Contributions

Our contributions are summarized as follows:

- **A Plonkish-compatible constraint system with almost-free additions.** We introduce a new constraint system that eliminates explicit addition gates by absorbing linear computation (in particular, additions) into the wiring relations of non-linear gates, while fully preserving support for high-degree custom gates.
- **A linear check protocol.** We design a new linear check, implemented via a single sumcheck, that enforces wiring consistency together with the absorbed linear relations in a unified manner.
- **The Pancake SNARK.** Building on the above, we construct Pancake, a SNARK with a linear-time prover under a circuit-specific trusted setup.
- **Implementation and evaluation.** We implement Pancake and benchmark it against HyperPlonk in both single-core and multi-core settings. Our results demonstrate consistent prover speedups and superior parallel scalability.

1.2 Technique overview

We begin by reviewing the constraint system of HyperPlonk [11], which serves as the foundation for our work. This review will help clarify our technical departure point and key innovation.

Consider an arithmetic circuit $\mathcal{C}$ with m fan-in-two gates, each performing an addition, multiplication, or a custom operation. Here, for simplicity, we assume that there is only one type of custom gate $\tilde{G} : \mathbb{F}^2 \to \mathbb{F}$, whereas in HyperPlonk more types of custom gates can be defined as needed. Let $\mathsf{x} \in \mathbb{F}^\ell$ be the public input. The wire values (left L_i, right R_i, output O_i) and public inputs are encoded in a matrix $\hat{T}$ with $n = m + \ell + 1 = 2^\mu$ rows:

$$\hat{T} = \{(L_i, R_i, O_i) \in \mathbb{F}^3\}_{i=0,\ldots,m+\ell}.$$

These values are interpolated into a *witness polynomial* $T : B_{\mu+2} \to \mathbb{F}$ over the Boolean hypercube, where $B_\mu := \{0,1\}^\mu$, such that for each row index i (with $\langle i \rangle_\mu$ being its μ-bit binary representation):

$$T(0,0,\langle i \rangle_\mu) = L_i, T(0,1,\langle i \rangle_\mu) = R_i, T(1,0,\langle i \rangle_\mu) = O_i,$$

The prover then demonstrates that T encodes a valid wire value assignment of the circuit. This process can be accomplished by satisfying two types of constraints: gate constraints and wiring constraints.

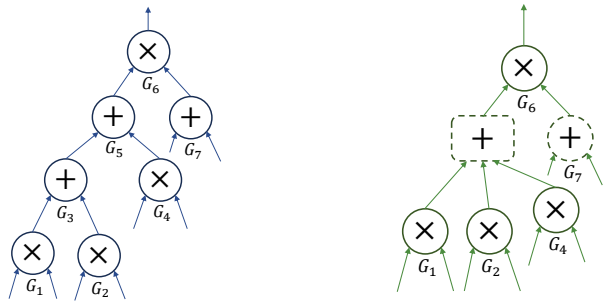
Gate constraints. Gate constraints ensure that each gate performs the correct arithmetic operation. For instance, an addition gate at index i must satisfy $L_i + R_i = O_i$, or equivalently $T(0,0,\langle i \rangle_\mu) + T(0,1,\langle i \rangle_\mu) = T(1,0,\langle i \rangle_\mu)$.

In our case, with three types of gates in the circuit, we need corresponding three multilinear selector polynomials $S_0, S_1, S_2 \in \mathbb{F}_\mu^{\leq 1}[\boldsymbol{X}]$, whose evaluations on B_μ lie in $\{0,1\}$, to identify the type of gate operation for each row of the matrix $\hat{T}$. The evaluations of the selector polynomials on $\langle i \rangle_\mu$ are specified in Table 1. It indicates that if gate i is an addition gate, then the evaluation of selector polynomials on $\langle i \rangle_\mu$ should follow the row with type "addition".

Table 1. Selector polynomial values and output constraints for different gate types.

Type	$S_0(\langle i \rangle_\mu)$	$S_1(\langle i \rangle_\mu)$	$S_2(\langle i \rangle_\mu)$	**Constraint**
Addition	1	0	0	$O_i = L_i + R_i$
Multiplication	0	1	0	$O_i = L_i \cdot R_i$
Custom	0	0	1	$O_i = \tilde{G}(L_i, R_i)$
Public Input	0	0	0	$O_i = \mathsf{PI}(\langle i \rangle_\mu)$

Let the multilinear polynomial $\mathsf{PI} : \mathbb{F}^\mu \to \mathbb{F}$ encode public inputs such that $\mathsf{PI}(\langle i \rangle_\mu) = \mathsf{x}_i$ for $i \in [0, \ell - 1]$ and is zero on the rest of B_μ. The prover demon-



(a) Original Plonkish (b) Ours

Fig. 2. Circuit representation comparison.

strates that the following polynomial vanishes (evaluates to zero) on all $\boldsymbol{x} \in B_\mu$:

$$0 = \mathsf{PI}(\boldsymbol{x}) + S_0(\boldsymbol{x}) \cdot \Big(T(0,0,\boldsymbol{x}) + T(0,1,\boldsymbol{x})\Big) + S_1(\boldsymbol{x}) \cdot T(0,0,\boldsymbol{x}) \cdot T(0,1,\boldsymbol{x}) + S_2(\boldsymbol{x}) \cdot \tilde{G}\Big(T(0,0,\boldsymbol{x}), T(0,1,\boldsymbol{x})\Big) - T(1,0,\boldsymbol{x}).$$

This is achieved via a *zerocheck*, a protocol which proves that a polynomial is identically zero on the Boolean hypercube, and can be reduced to a sumcheck.

Wiring constraints. Wiring constraints ensure the correct propagation of wire values through a circuit. Consider the circuit in Fig. 2a, which contains 3 addition gates (G_3, G_5, G_7) and 4 multiplication gates (G_1, G_2, G_4, G_6). The wiring constraints enforce the following equalities:

$$O_1 = L_3, O_2 = R_3, O_3 = L_5, O_4 = R_5, O_5 = L_6, O_7 = R_6.$$

In terms of the witness polynomial T, this translates to ensuring:

$$T(1,0,\langle 1\rangle_3) = T(0,0,\langle 3\rangle_3), \quad T(1,0,\langle 2\rangle_3) = T(0,1,\langle 3\rangle_3),$$
$$T(1,0,\langle 3\rangle_3) = T(0,0,\langle 5\rangle_3), \quad T(1,0,\langle 4\rangle_3) = T(0,1,\langle 5\rangle_3),$$
$$T(1,0,\langle 5\rangle_3) = T(0,0,\langle 6\rangle_3), \quad T(1,0,\langle 7\rangle_3) = T(0,1,\langle 6\rangle_3).$$

For a circuit with w wiring constraints, each constraint $t \in [0, w-1]$ is defined by a pair $(\boldsymbol{b}_0^{(t)}, \boldsymbol{b}_1^{(t)})$ requiring $T(\boldsymbol{b}_0^{(t)}) = T(\boldsymbol{b}_1^{(t)})$.

To enforce the wiring constraints, HyperPlonk utilizes a permutation check. First, a permutation $\sigma : B_{\mu+2} \to B_{\mu+2}$ is defined to encode all wiring relations: for each wiring pair $(\boldsymbol{b}_0^{(t)}, \boldsymbol{b}_1^{(t)})$, it sets $\sigma(\boldsymbol{b}_0^{(t)}) = \boldsymbol{b}_1^{(t)}$. For wires that are not part of the wiring relations, their permutations simply map to themselves.

The prover needs to demonstrate that $T(\boldsymbol{b}) = T(\sigma(\boldsymbol{b}))$ for all $\boldsymbol{b} \in B_{\mu+2}$. To achieve this, HyperPlonk transforms the problem into a multiset equality check:

prove that $\{(\boldsymbol{b}, T(\boldsymbol{b}))\}_{\boldsymbol{b}\in B_{\mu+2}}$ and $\{(\sigma(\boldsymbol{b}), T(\boldsymbol{b}))\}_{\boldsymbol{b}\in B_{\mu+2}}$ are equal. This multiset equality is established using a product argument: given random challenges r_1, r_2, r_3 from the verifier, the prover shows that

$$\prod_{\boldsymbol{b}\in B_{\mu+2}} \big(r_1 + r_2 \cdot \boldsymbol{b} + r_3 \cdot T(\boldsymbol{b})\big) = \prod_{\boldsymbol{b}\in B_{\mu+2}} \Big(r_1 + r_2 \cdot \sigma(\boldsymbol{b}) + r_3 \cdot T\big(\boldsymbol{b}\big)\Big)$$

Equivalently, defining $f_1(\boldsymbol{X}) = r_1 + r_2 \cdot \boldsymbol{X} + r_3 \cdot T(\boldsymbol{X})$, $f_2(\boldsymbol{X}) = r_1 + r_2 \cdot \sigma(\boldsymbol{X}) + r_3 \cdot T(\boldsymbol{X})$, and $f'(\boldsymbol{X}) = f_1(\boldsymbol{X})/f_2(\boldsymbol{X})$, the prover shows that the accumulated product of f' over $B_{\mu+2}$ equals 1. This productcheck protocol is subsequently reduced (together with the gate constraints) to a zerocheck, and then to a sumcheck, as illustrated in Fig. 3b.

New approach for proving wiring constraints. Instead of the permutation check used in HyperPlonk, Pancake proves wiring constraints directly via a single sumcheck. The key idea is to batch all wiring equalities into one linear relation using random coefficients.

Consider two wiring constraints: $T(\boldsymbol{a}_0) = T(\boldsymbol{a}_1)$ and $T(\boldsymbol{a}_1) = T(\boldsymbol{a}_2)$. Given a random challenge $r \in \mathbb{F}$ from the verifier, we can combine them into:

$$r \cdot T(\boldsymbol{a}_0) - r \cdot T(\boldsymbol{a}_1) + r^2 \cdot T(\boldsymbol{a}_1) - r^2 \cdot T(\boldsymbol{a}_2) = 0.$$

This technique generalizes directly: for a set of w wiring constraints, each encoded as a pair $\mathcal{Q}^{(t)} := (\boldsymbol{b}_0^{(t)}, \boldsymbol{b}_1^{(t)})$, where $t \in [0, w-1]$. We have

$$\textstyle\sum_{t=0}^{w-1} r^t \cdot \big(T(\boldsymbol{b}_0^{(t)}) - T(\boldsymbol{b}_1^{(t)})\big) = 0. \tag{1}$$

To verify Equation (1) efficiently, we encode the assigned random values of wiring relation into a selector polynomial $W_r : B_{\mu+2} \to \mathbb{F}$. We first build a coefficient vector $\mathbf{w} \in \mathbb{F}^{2^{\mu+2}}$ initialized to zero. For each constraint t, we update the entries corresponding to the wire indices:

$$\mathbf{w}[\boldsymbol{b}_0^{(t)}] \leftarrow \mathbf{w}[\boldsymbol{b}_0^{(t)}] + r^t, \quad \mathbf{w}[\boldsymbol{b}_1^{(t)}] \leftarrow \mathbf{w}[\boldsymbol{b}_1^{(t)}] - r^t.$$

Define W_r such that $W_r(\boldsymbol{b}) = \mathbf{w}[\boldsymbol{b}]$ for all $\boldsymbol{b} \in B_{\mu+2}$. Then, Equation (1) is equivalent to:

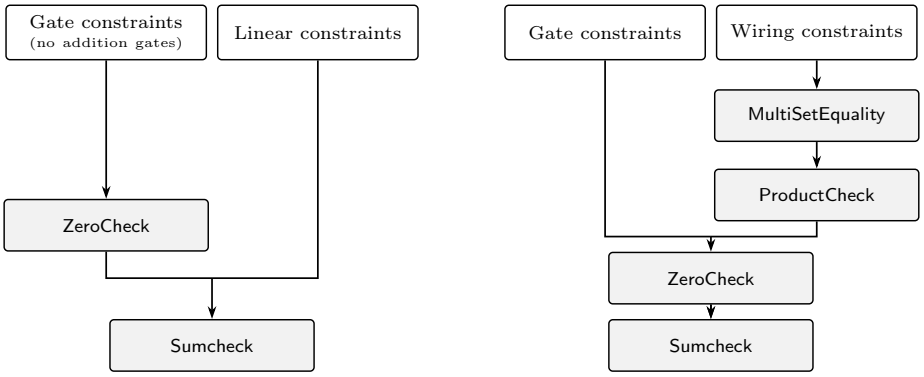
$$\textstyle\sum_{\boldsymbol{x}\in B_{\mu+2}} W_r(\boldsymbol{x}) \cdot T(\boldsymbol{x}) = 0,$$

which is directly amenable to a single sumcheck protocol.

New linear constraints. Our core technical observation is that both wiring constraints and addition gate constraints are linear relations over the wire values. This allows them to be unified into a single set of *linear constraints*.

A wiring constraint enforces the equality of two wire values, e.g., $T(\boldsymbol{a}) = T(\boldsymbol{b})$, which is equivalent to $T(\boldsymbol{a}) - T(\boldsymbol{b}) = 0$. An addition gate constraint, such as $L_i + R_i - O_i = 0$, is inherently linear. Consequently, both can be captured in a common form:

$$\textstyle\sum_j \beta_j \cdot T(\boldsymbol{b}_j) = 0,$$



(a) PIOPs in Pancake (b) PIOPs in HyperPlonk

Fig. 3. The multilinear PIOPs underlying Pancake and HyperPlonk.

where β_j are scalar coefficients and $\boldsymbol{b}_j$ are hypercube points identifying specific wires. Formally, each constraint t is described by a tuple:

$$\mathcal{Q}^{(t)} := ((\beta_0^{(t)}, \boldsymbol{b}_0^{(t)}), (\beta_1^{(t)}, \boldsymbol{b}_1^{(t)}), \dots).$$

For instance, the linear relation $T(\boldsymbol{a}_1) + 4 \cdot T(\boldsymbol{a}_2) - T(\boldsymbol{a}_3) = 0$ corresponds to the tuple $(1, \boldsymbol{a}_1, 4, \boldsymbol{a}_2, -1, \boldsymbol{a}_3)$.

Critically, this unification transforms how addition gates are verified. In our framework, addition gates are no longer checked by explicit gate constraints. Instead, each is expressed as a linear relation among its input and output wires. By substituting the wiring connections into these relations, the intermediate wires belonging to addition gates are "algebraically eliminated." What remains is a new set of constraints that link only the wires of non-addition gates (i.e., multiplication and custom gates). These consolidated constraints simultaneously enforce correct circuit wiring and the correctness of all original addition operations.

This substitution capability also allows a single linear relation to encode the constraints of multiple addition gates. For example, consider Fig. 2a, where verifying addition gates G_3 and G_5 traditionally requires seven constraints: $L_3 + R_3 = O_3$, $L_5 + R_5 = O_5$, $O_1 = L_3$, $O_2 = R_3$, $O_3 = L_5$, $O_4 = R_5$, $O_5 = L_6$. In our approach, we absorb these into a single linear relation that involves only multiplication gate wires: $O_1 + O_2 + O_4 = L_6$, which implicitly ensures the correctness of G_3 and G_5 as shown in Fig. 2b. Consequently, the total number of linear constraints w is typically much smaller than the original count of addition gates and wiring pairs.

Since addition gates are verified implicitly through the linear constraints, the witness polynomial T now only needs to encode the $N = 2^\nu$ non-addition gates. This reduces its domain size from 2^μ (all gates) to 2^ν, lowering the cost of all subsequent polynomial operations. The gate constraint is also simplified to only

check non-addition gates, where the prover shows that for all $\boldsymbol{x} \in B_\nu$:

$$0 = \mathsf{PI}(\boldsymbol{x}) + S_1(\boldsymbol{x}) \cdot \Big(T(0,0,\boldsymbol{x}) \cdot T(0,1,\boldsymbol{x})\Big) + S_2(\boldsymbol{x}) \cdot \tilde{G}\Big(T(0,0,\boldsymbol{x}), T(0,1,\boldsymbol{x})\Big) - T(1,0,\boldsymbol{x}).$$

As shown in Fig. 3a, the constraint system of the Pancake comprises two main components: general linear constraints and gate constraints, with the latter only defining non-addition gate operations.

Putting it all together. For each constraint $t \in [0, w-1]$, we show: $\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot T(\boldsymbol{b}_i^{(t)}) = 0$. With a random verifier challenge r, we batch these into a single equation:

$$\sum_{t=0}^{w-1} r^t \left(\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot T(\boldsymbol{b}_i^{(t)})\right) = 0. \tag{2}$$

To transform Equation (2) into a sumcheck, we first encode the coefficients of each constraint t into a multilinear polynomial H_t as

$$H_t(\boldsymbol{X}) := \sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot \mathsf{eq}(\boldsymbol{b}_i^{(t)}, \boldsymbol{X}),$$

where $\mathsf{eq}(\boldsymbol{b}, X) := \prod_{i=1}^{\nu+2} \big(b_i X_i + (1-b_i)(1-X_i)\big)$, and $\mathsf{eq}(\boldsymbol{b}, \boldsymbol{x}) = 1$ if $\boldsymbol{b} = \boldsymbol{x}$, and 0 otherwise. We assume w.l.o.g. that the points $\boldsymbol{b}_i^{(t)}$ within each constraint are pairwise distinct (repeated points are merged by summing their coefficients). Thus, by construction, $H_t(\boldsymbol{b}_i^{(t)}) = \beta_i^{(t)}$ and $H_t(\boldsymbol{x}) = 0$ for all other $\boldsymbol{x} \in B_{\nu+2}$. This allows us to rewrite each constraint as a sum over $B_{\nu+2}$ as

$$\begin{aligned}\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot T(\boldsymbol{b}_i^{(t)}) &= \sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} H_t(\boldsymbol{b}_i^{(t)}) \cdot T(\boldsymbol{b}_i^{(t)}) \\ &= \sum_{\boldsymbol{x} \in B_{\nu+2}} H_t(\boldsymbol{x}) T(\boldsymbol{x}).\end{aligned}$$

Substituting this into the batched Equation (2) yields:

$$\sum_{t=0}^{w-1} r^t \left(\sum_{x \in B_{\nu+2}} H_t(\boldsymbol{x}) \cdot T(\boldsymbol{x})\right) = 0$$

This can also be expressed as:

$$\sum_{x \in B_{\nu+2}} \sum_{t=0}^{w-1} r^t H_t(\boldsymbol{x}) \cdot T(\boldsymbol{x}) = 0.$$

We define the selector polynomial W_r (overloading the wiring-only selector introduced above) as the inner sum, which aggregates all batched coefficients:

$$W_r(\boldsymbol{X}) := \sum_{t=0}^{w-1} r^t \cdot H_t(\boldsymbol{X}).$$

This allows us to express the entire set of linear constraints as a single sumcheck

$$\sum_{\boldsymbol{x} \in B_{\nu+2}} W_r(\boldsymbol{x}) \cdot T(\boldsymbol{x}) = 0. \tag{3}$$

Challenge: Efficient evaluation of W_r. In the final round of sumcheck protocol (see section 3.4) for Equation (3), the verifier needs to evaluate W_r at a random point $\boldsymbol{\alpha}$. Direct computation of $W_r(\boldsymbol{\alpha})$ by the verifier incurs linear time complexity, which conflicts with our goal of achieving succinct verification.

A natural idea is to precompute W_r and commit to it during the offline phase, so that evaluation at any point can be handled efficiently online. However, constructing W_r depends on the random challenge r, which is chosen in the online phase. This dependency prevents fully precomputing W_r offline.

To address this, we introduce a new polynomial $W(\boldsymbol{X}, Y)$, which can be constructed offline, by replacing the random scalar r in $W_r(\boldsymbol{X})$ with a formal variable Y:

$$W(\boldsymbol{X}, Y) := \sum_{t=0}^{w-1} Y^t \cdot H_t(\boldsymbol{X}).$$

The verifier can store the commitment C_W of W. With commitment C_W in hand, in the online phase, the verifier now asks the prover to open $W(\boldsymbol{\alpha}, r)$. However, directly opening the commitment C_W at the point $(\boldsymbol{\alpha}, r)$ would be computationally expensive for typical multilinear polynomial commitment schemes, since W is a $(\nu + 3)$-variate polynomial with degree up to $w - 1$ in Y.

Instead of a direct opening on $(\boldsymbol{\alpha}, r)$, we exploit the relation $W_r(\boldsymbol{x}) = W(\boldsymbol{x}, r)$ and require prover to first open $W(\boldsymbol{X}, Y)$ on $Y = r$. This is achieved by proving the existence of a quotient polynomial $Q(\boldsymbol{X}, Y)$ that satisfies the identity

$$Q(\boldsymbol{X}, Y)(Y - r) = W(\boldsymbol{X}, Y) - W_r(\boldsymbol{X}) \tag{4}$$

If Equation (4) holds, then evaluating it at $Y = r$ and $\boldsymbol{X} = \boldsymbol{\alpha}$ gives $W(\boldsymbol{\alpha}, r) = W_r(\boldsymbol{\alpha})$. The prover therefore opens $W_r(\boldsymbol{X})$ at $\boldsymbol{\alpha}$, which returns the value $W(\boldsymbol{\alpha}, r)$.

Challenge: Costly online computation of Q. Our construction leverages the multilinear KZG polynomial commitment scheme [36] (see Section 3.5), which is additively homomorphic. With KZG, we can check Equation (4) using the pairing-based relation involving commitments to Q, W, and W_r. However, naively committing to Q is still very expensive.

We address this inefficiency by expressing Q as a linear combination of basis polynomials p_i, for which the commitments can be precomputed offline. As a result, the commitment to Q can be efficiently computed in linear time online.

We notice that the valid quotient polynomial Q from Equation (4) has the following structure

$$Q(\boldsymbol{X}, Y) = \sum_{t=0}^{w-1} \frac{Y^t - r^t}{Y - r} \cdot H_t(\boldsymbol{X}).$$

With the algebraic identity $\frac{Y^t - r^t}{Y - r} = \sum_{k=0}^{t-1} r^{t-1-k} \cdot Y^k$, we rewrite Q as

$$Q(\boldsymbol{X}, Y) = \sum_{t=1}^{w-1} \left(\sum_{k=0}^{t-1} r^{t-1-k} \cdot Y^k \right) \cdot H_t(\boldsymbol{X}).$$

Rearranging to isolate powers of the verifier challenge r yields

$$Q(\boldsymbol{X}, Y) = \sum_{i=0}^{w-2} r^i \cdot \left(\sum_{t=i+1}^{w-1} Y^{t-i-1} \cdot H_t(\boldsymbol{X}) \right)$$

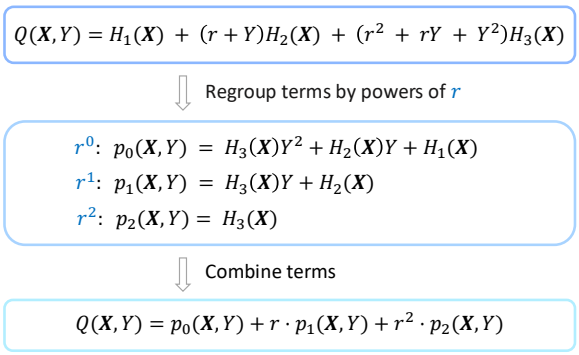


Fig. 4. Example of rearranging terms in quotient polynomial Q

Now we define a set of basis polynomials p_i for all $i \in [0, w-2]$ as

$$p_i(\boldsymbol{X}, Y) := \sum_{t=i+1}^{w-1} Y^{t-i-1} \cdot H_t(\boldsymbol{X}). \tag{5}$$

This allows Q to be expressed as a linear combination of these p_i polynomials, with coefficients being powers of r:

$$Q(\boldsymbol{X}, Y) = \sum_{i=0}^{w-2} r^i \cdot p_i(\boldsymbol{X}, Y).$$

For example, if there were $w = 4$ constraints, we present the decomposition for Q in Fig. 4.

Since the commitments C_i to p_i can be precomputed offline, the prover can compute the commitment C_Q online as $C_Q := \sum_{i=0}^{w-2} r^i \cdot C_i$. This calculation requires only $O(w)$ group operations, avoiding both explicit polynomial division and the heavy cost of naive commitment.

The efficiency of this approach relies on precomputing the commitments C_W and C_i. Using a generic universal setup to commit to these polynomials would involve $O(w \cdot 2^\nu)$ group operations, which is prohibitively expensive. In contrast, our scheme utilizes a linear-time circuit-specific setup. By exploiting the known support pattern and recursive structure of H_t, we can generate all necessary commitments in linear time.

2 Related Work

Achieving linear prover time has become a central focus in recent SNARK research. The Spartan framework [39] provides a foundational paradigm in this direction: it encodes R1CS instances as multilinear polynomials to circumvent FFTs, and leverages the sparsity of constraint matrices to achieve linear-time proving. Building upon this multivariate foundation, several works have explored different trade-offs. Quarks [41] advances this line by instantiating Spartan with highly optimized polynomial commitment schemes, yielding transparent

SNARKs that combine linear-time proving with sublinear verification. A separate thread of work focuses on eliminating field-specific restrictions and achieving post-quantum security. Lee et al. [32] and Brakedown [20] instantiate the Spartan framework with linear-time error-correcting codes, producing field-agnostic SNARKs with strictly linear-time provers. This approach, however, inherently comes at the cost of larger proof sizes (e.g., $O(\sqrt{n})$), both asymptotically and concretely. Testudo [10] maintains Spartan's linear-time prover while achieving constant-size proofs and a square-root-size universal setup; it optimizes verification by delegating heavy proof verification to a Groth16 prover, though its handling of non-uniform circuits incurs some overhead. Most recently, Samaritan [16] integrates the Log-Up lookup argument to directly reduce Spartan's transparent proof size from $O(\log^2 n)$ to $O(\log n)$, demonstrating how modern lookup techniques can further compress Spartan-style protocols.

Beyond the Spartan framework, alternative constructions pursue linear-time proving through different paradigms. LegoUAC [9] employs a modular commit-and-prove framework based on matrix product arguments, achieving linear prover time with polylogarithmic proof sizes. Poppins [31] introduces a constraint system tailored for linear-time proving, though its concrete prover cost is reportedly higher than several earlier SNARKs. Gemini [7] proposes elastic SNARKs to offer a flexible time-space trade-off: a time-efficient mode with linear prover time, and a space-efficient mode that streams the computation with logarithmic memory usage.

A significant advancement is HyperPlonk [11], which adapts Plonk [15] to the Boolean hypercube using multilinear polynomials, achieving linear prover complexity while preserving support for custom gates. Pancake builds upon this foundation but departs from its constraint formulation. Instead of expressing linear relations through explicit addition gates, Pancake reformulates the constraint system to absorb both linear operations and wiring consistency into a unified set of linear constraints, effectively rendering addition gates "almost free".

In the realm of circuit-specific setups, Groth16 [23] remains the industry benchmark for succinctness (three group elements), albeit requiring $O(n \log n)$ prover time due to its reliance on FFTs. Recently, Garuda [14] proposed a linear-time SNARK that, like Pancake, leverages a circuit-specific trusted setup to make linear operations essentially free and to support high-degree custom gates. The fundamental architectural distinction lies in their constraint representations: Garuda operates over the Generalized R1CS (GR1CS) framework, while Pancake is built upon the Plonkish model. Because Pancake retains the Plonkish gate model, it directly accommodates high-degree custom gates already expressed in that model, whereas Garuda represents the computation through GR1CS.

Finally, we note similarities with Dynamo [48], a sparse zk-SNARK for permutation relations by employing random linear combinations technique. The key distinction lies in the algebraic encoding: Dynamo relies on univariate Lagrange polynomials, which, when extended to the multivariate setting, necessitate a higher-dimensional sumcheck involving twice the number of variables (i.e., 2μ variables). In contrast, Pancake utilizes a monomial basis (Y^i) to batch linear

relations, which preserves the original variable count and results in a simpler multivariate sumcheck.

3 Preliminaries

We use $\mathbb{F}$ to denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a large prime p) and $\lambda \in \mathbb{N}$ to denote the security parameter. We denote the Boolean hypercube by $B_\mu := \{0,1\}^\mu$.

We use $[n]$ to denote the integer set $\{1, 2, \ldots, n\}$, and $[n, m]$ to denote $\{n, n+1, \ldots, m\}$ for integers $m > n$. For a vector $\boldsymbol{f}$, which is written in bold, we use f_i to denote the i-th element in $\boldsymbol{f}$. For a finite set S, we use $s \xleftarrow{\$} S$ to denote that an element s is sampled uniformly at random from set S. For any $m \in \mathbb{N}$ and $i \in [0, 2^m)$, we use $\langle i \rangle_m = v \in B_m$ to denote the m-bit binary representation of i, that is, $i = \sum_{j=1}^{m} v_j \cdot 2^{j-1}$.

Indexed relation An indexed relation $\mathcal{R}$ is a set of tuples $(\mathbb{i}; \mathbb{x}; \mathbb{w})$, where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, $\mathbb{w}$ is the witness. The corresponding indexed language $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}; \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}; \mathbb{x}; \mathbb{w}) \in \mathcal{R}$. Given an indexed relation $\mathcal{R}$, the corresponding binary relation can be defined as $\mathcal{R}_b = \{((\mathbb{i}, \mathbb{x}); \mathbb{w}) : (\mathbb{i}; \mathbb{x}; \mathbb{w}) \in \mathcal{R}\}$. Typically, $\mathbb{i}$ describes an arithmetic circuit over a finite field $\mathbb{F}$, $\mathbb{x}$ denotes public inputs, and $\mathbb{w}$ denotes private witness.

Pairings We consider bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \mathsf{g}, \mathsf{h})$ characterized by the following properties: $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are groups of prime order p, and there exists a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The elements g and h are generators for $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, with $e(\mathsf{g}, \mathsf{h})$ generating $\mathbb{G}_T$. Additionally, there are efficient algorithms for performing group operations, evaluating the bilinear map, verifying group membership, checking equality of elements, and sampling generators.

We write $[a]_1$ for g^a, $[b]_2$ for h^b. Under this convention, we have generators $\mathsf{g} = [1]_1$ and $\mathsf{h} = [1]_2$. Throughout, we use additive notation for group operations. For example, in this notation, $[a]_1 + [b]_1 = [a + b]_1$.

Definition 1. *A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.*

We use $\mathbb{F}_\mu^{\leq d}[\boldsymbol{X}]$ to denote the set of multivariate polynomials in $\mathbb{F}[X_1, \ldots, X_\mu]$ where the degree in each variable is at most d;

Lemma 1 (Multilinear extensions (MLE)). *For every function $f : B_\mu \to \mathbb{F}$ that maps μ-bit elements to elements of $\mathbb{F}$, there is a unique multilinear polynomial $\tilde{f} \in \mathbb{F}_\mu[\boldsymbol{X}]$ such that $\tilde{f}(\boldsymbol{b}) = f(\boldsymbol{b})$ for all $\boldsymbol{b} \in B_\mu$. We call $\tilde{f}$ the* multilinear extension *(MLE) of f, and $\tilde{f}$ can be expressed as:*

$$\tilde{f}(X) = \textstyle\sum_{\boldsymbol{b} \in B_\mu} f(\boldsymbol{b}) \cdot \mathsf{eq}(\boldsymbol{b}, X),$$

where $\mathsf{eq}(\boldsymbol{b}, X) := \prod_{i=1}^{\mu} \big(b_i X_i + (1-b_i)(1-X_i)\big)$. *For any vector* $\boldsymbol{b} \in B_\mu$, $\mathsf{eq}(\boldsymbol{b}, X)$ *satisfies* $\mathsf{eq}(\boldsymbol{b}, \boldsymbol{b}) = 1$ *and* $\mathsf{eq}(\boldsymbol{b}, \boldsymbol{b}') = 0$ *for any* $\boldsymbol{b}' \neq \boldsymbol{b}$.

3.1 Arguments of knowledge

An interactive argument of knowledge for an NP relation $\mathcal{R}$ is a tuple of algorithms $\mathit{\Pi} = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$. Setup generates the proving key pp and verification key vp. $\mathcal{P}$ is a computationally-bounded prover who interacts with a verifier $\mathcal{V}$ to convince the verifier that there exists a witness $\mathbb{w}$ such that $(\mathbb{x}; \mathbb{w}) \in \mathcal{R}$ for a public statement $\mathbb{x}$ through rounds of interaction. The interactive protocol Π is an argument of knowledge for $\mathcal{R}$ if it satisfies perfect completeness and knowledge soundness. We adopt an existing definition [23]

- **Perfect Completeness:** For all $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\Pr[\langle \mathcal{P}(\mathsf{pp}, \mathbb{w}), \mathcal{V}(\mathsf{vp}) \rangle(\mathbb{x}) = 1] = 1$. Here, $\langle \mathcal{P}, \mathcal{V} \rangle(\mathbb{x})$ denotes the final decision of the verifier $\mathcal{V}$ after interacting with $\mathcal{P}$ on common input $\mathbb{x}$.
- **Knowledge Soundness:** For any $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{Setup}(1^\lambda)$, any statement $\mathbb{x}$, and any PPT prover $\mathcal{P}^*$, there exists an expected polynomial-time extractor E with oracle access to $\mathcal{P}^*$ such that $\Pr[\langle \mathcal{P}^*(\mathsf{pp}), \mathcal{V}(\mathsf{vp}) \rangle(\mathbb{x}) = 1, (\mathbb{x}, \mathbb{w}) \notin \mathcal{R} \mid \mathbb{w} \leftarrow E^{\mathcal{P}^*(\mathsf{pp})}(\mathbb{x})] \leq \mathrm{negl}(\lambda)$.

Succinct Non-interactive Arguments of Knowledge (SNARKs) Interactive public-coin arguments can be transformed into non-interactive arguments using the Fiat-Shamir transform. This transformation replaces the verifier challenges with hashes of the transcript so far. Several works [1,49] show that this is secure for multi-round special-sound protocols and multi-round oracle proofs.

A protocol is considered public coin if all of the verifier's messages can be computed deterministically from a random public input. A non-interactive argument of knowledge (NARK) is considered succinct if, for every $(\mathbb{x}; \mathbb{w}) \in \mathcal{R}$, the proof size is $\mathrm{poly}(\lambda, \log(|\mathbb{x}| + |\mathbb{w}|))$.

3.2 PIOP

SNARKs can be constructed based on a combination of a cryptographic commitment scheme and an information-theoretic proof system, such as a polynomial commitment scheme and a polynomial interactive oracle proof (PIOP). In an interactive oracle proof (IOP) [3], each message is a string. Instead of reading the entire prover's messages, the verifier $\mathcal{V}$ has oracle access to these messages, meaning that the verifier can query individual symbols of the message strings via random access. A PIOP [8] is a specific type of IOP in which the prover $\mathcal{P}$ encodes its messages as polynomials and provides oracle access to these polynomials to $\mathcal{V}$. The verifier $\mathcal{V}$ can then query the oracles by requesting the evaluations of the polynomials at randomly chosen points.

Definition 2 (PIOP). *[11] Let* $\mathcal{R} = \{(\mathbb{i}; \mathbb{x}; \mathbb{w})\}$ *be a polynomial oracle relation over* $\mathbb{F}$*, where* $\mathbb{i}$ *and* $\mathbb{x}$ *can contain oracles to* μ*-variate polynomials over some*

field $\mathbb{F}$. The oracles specify μ and the degree of each variable. The oracles allow evaluation of the polynomials at arbitrary points in $\mathbb{F}^\mu$. The actual polynomials that these oracles represent are contained within the pp *and the witness* $\mathbb{w}$*. We denote an oracle to a polynomial f as $[[f]]$. In every round, the prover sends the multivariate polynomial oracles to the verifier, and the verifier then responds with random challenges, where the challenges are used to derive the query points to the oracles. Upon receiving the evaluations of the query points from the prover, the verifier outputs a decision bit.*

Proof of Knowledge Typically, PIOP satisfies perfect completeness, and δ-knowledge-soundness. A formal and complete definition is provided in [8,11].

Complexity measures We measure the complexity for the PIOP based on the following parameters:

- *Query complexity*: number of queries the verifier performs to the oracles.
- *Round complexity*: number of rounds in the protocol, which is also the number of sent oracles.
- *Proof oracle size*: length of the transmitted polynomials.
- *Prover time*: time complexity of the prover.
- *Verifier time*: time complexity of the verifier.

Virtual Oracles and Commitments

Given a set of polynomial oracles $[[f_1]], \ldots, [[f_k]]$, we define a virtual oracle for a function $g(f_1, \ldots, f_k)$ as $\{[[f_1]], \ldots, [[f_k]]\}$ and a description of g. To evaluate this virtual oracle at a point $\boldsymbol{x}$, we first query each oracle to obtain $y_i = f_i(\boldsymbol{x})$ for all $i \in [k]$, and then compute and return $g(y_1, \ldots, y_k)$.

This concept extends naturally to polynomial commitments. A virtual commitment to $g([[f_1]], \ldots, [[f_k]])$ consists of the individual commitments and the function description. If the underlying commitment scheme is additively homomorphic, we can further exploit this property whenever g is an additive function. For instance, given commitments C_f and C_g to polynomials f and g respectively, a commitment to the linear combination $(1-Y)f+Yg$ can be represented as the pair (C_f, C_g). To evaluate this at a point (y, x), we compute $(1-y)C_f + y \cdot C_g$ evaluated at x, utilizing the scheme's additive homomorphism.

3.3 PIOP Compilation

A PIOP protocol Π_O can be compiled into interactive arguments of knowledge Π_A by leveraging cryptographic techniques such as polynomial commitments. Given a polynomial commitment scheme $\Pi_{\mathsf{PC}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$. In the compilation process, for each oracle polynomial in the PIOP, the prover sends its commitment. Subsequently, every oracle query by the verifier to a polynomial at some points is replaced with an invocation of the PCS's Eval algorithm at that point [8,12,11]. The efficiency of Π_A depends on the efficiency of both the PIOP Π_O and Π_{PC}.

Witness-extended emulation The property of witness-extended emulation strengthens the knowledge-soundness by requiring the extractor to produce both a witness and a simulated transcript between the prover and verifier [33]. A formal and complete definition is provided in [8].

Theorem 1 (PIOP Compilation [12,8]). *Let Π_{PC} be a polynomial commitment scheme and Π_O be a t-round Polynomial IOP for a relation $\mathcal{R}$ over a field $\mathbb{F}$. If Π_{PC} has witness-extended emulation and Π_O has negligible knowledge error, then the PIOP compilation protocol Π_S is a secure public-coin interactive argument of knowledge for $\mathcal{R}$.*

The proof of Theorem 1 is presented in [8].

3.4 PIOP building blocks

Sumcheck We recall the sumcheck PIOP used in HyperPlonk [11], which follows the classical sumcheck protocol [34,46].

Definition 3 (Sumcheck relation). *The relation $\mathcal{R}_{\mathsf{sum}}$ consists of tuples $(\mathbb{x}; \mathbb{w}) = ((v, [[f]]); f)$, where $f \in \mathbb{F}_{\mu}^{\leq d}[\boldsymbol{X}]$ satisfies $\sum_{\boldsymbol{b} \in B_\mu} f(\boldsymbol{b}) = v$.*

The protocol Π_{sum} for $\mathcal{R}_{\mathsf{sum}}$ is described below.

Protocol Π_{sum}. $\mathcal{P}(\mathbb{x}, \mathbb{w})$ and $\mathcal{V}(\mathbb{x})$ run the following:
For $i = \mu, \mu - 1, \ldots, 1$:

- The prover computes the univariate polynomial

$$t_i(X) := \textstyle\sum_{\boldsymbol{b} \in B_{i-1}} f(\boldsymbol{b}, X, \alpha_{i+1}, \ldots, \alpha_\mu),$$

  and derives the polynomial:

$$r_i(X) := \tfrac{t_i(X) - (1-X)\cdot t_i(0) - X \cdot t_i(1)}{X \cdot (1-X)}.$$

  The prover sends the oracle $[[r_i]]$ and the evaluation $t_i(0)$ to the verifier, where r_i is univariate and of degree at most $d - 2$.
- The verifier computes $t_i(1) := v - t_i(0)$, samples $\alpha_i \xleftarrow{\$} \mathbb{F}$ and queries $r_i(\alpha_i)$.
- The verifier computes

$$t_i(\alpha_i) = r_i(\alpha_i) \cdot (1 - \alpha_i) \cdot \alpha_i + (1 - \alpha_i) \cdot t_i(0) + \alpha_i \cdot t_i(1).$$

  The verifier updates $v \leftarrow t_i(\alpha_i)$.

Finally, the verifier accepts if $f(\alpha_1, \ldots, \alpha_\mu) = v$.

Fig. 5. Sumcheck PIOP Π_{sum} for $\mathcal{R}_{\mathsf{sum}}$.

For a multivariate polynomial $f(\boldsymbol{X}) = h(g_1(\boldsymbol{X}), \ldots, g_\kappa(\boldsymbol{X}))$, where polynomials $g_i \in \mathbb{F}^{\leq 1}[\boldsymbol{X}]$ for $i \in [\kappa]$ are multilinear, and h is a c-variate polynomial

of total degree d and is computable via an arithmetic circuit with $O(d)$ gates, Hyperplonk leverages techniques from [45,50]. These methods enable the prover to compute each round's univariate polynomial r_i in $O(2^\mu \cdot d \log^2 d)$ field operations using dynamic programming, provided f has low circuit depth that can be evaluated in $O(d)$ time.

Theorem 2. *The PIOP* Π_{sum} *for relation* $\mathcal{R}_{\mathsf{sum}}$ *has query complexity* $\mu + 1$*, round complexity* μ*, proof oracle size* $d \cdot \mu$*, prover time* $O(2^\mu \cdot d \log^2 d)$*, and verifier time* $O(\mu)$*.* Π_{sum} *is perfectly complete and has knowledge error of* $\frac{d\mu}{|\mathbb{F}|}$*.*

For the proof, we refer to [46,11].

Zerocheck We now describe a PIOP that proves a multivariate polynomial evaluates to zero at every point in the Boolean hypercube. First, we define the ZeroCheck relation $\mathcal{R}_{\mathsf{zero}}$ as follows:

Definition 4 (ZeroCheck relation). *The zerocheck relation* $\mathcal{R}_{\mathsf{zero}}$ *is the set of tuples* $(\mathbb{x}; \mathbb{w}) = ([[f]]; f)$*, where* $f \in \mathbb{F}_\mu^{\leq d}[\boldsymbol{X}]$ *and* $f(\boldsymbol{x}) = 0$ *for all* $\boldsymbol{x} \in B_\mu$*.*

We describe the PIOP for proving $\mathcal{R}_{\mathsf{zero}}$ from [11,39] as in Fig. 6.

Protocol Π_{zero}**.** $\mathcal{P}(\mathbb{x}, \mathbb{w})$ and $\mathcal{V}(\mathbb{x})$ run the following:

1. $\mathcal{V}$ sends $\mathcal{P}$ a random vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^\mu$.
2. Let $\hat{f}(\boldsymbol{X}) = f(\boldsymbol{X}) \cdot \mathsf{eq}(\boldsymbol{X}, \boldsymbol{r})$.
3. Run the Sumcheck PIOP Π_{sum} for $(0, [[\hat{f}]]; \hat{f}) \in \mathcal{R}_{\mathsf{sum}}$.

Fig. 6. Zerocheck PIOP Π_{zero} for R_{zero}.

Theorem 3. *The PIOP* Π_{zero} *for relation* $\mathcal{R}_{\mathsf{zero}}$ *has query complexity* $\mu + 1$*, round complexity* μ*, proof oracle size* $d \cdot \mu$*, prover time* $O(2^\mu \cdot d \log^2 d)$*, and verifier time* $O(\mu)$*.* Π_{zero} *is perfectly complete and has knowledge error of* $O(\frac{d\mu}{|\mathbb{F}|})$*.*

We refer to [11] (Section 3.2) for the proof.

3.5 Multilinear polynomial commitment scheme

A polynomial commitment scheme (PCS) [29] enables a sender to commit to a polynomial while preserving the ability to later prove specific evaluations of that polynomial efficiently. We describe the KZG-based [29] multilinear polynomial commitment scheme [36] as follows.

- $\mathsf{crs} \leftarrow \mathsf{KZG.Setup}(1^\lambda, \mu)$: On input of a security parameter λ, and the number of variables μ. Pick $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_\mu) \xleftarrow{\$} \mathbb{F}^\mu$. Let $U := \{[\mathsf{eq}(\boldsymbol{b}, \boldsymbol{\tau})]_1\}_{\boldsymbol{b} \in B_\mu}$. Output $\mathsf{crs} = \{U, [1]_1, [1]_2, [\tau_1]_2, \ldots, [\tau_\mu]_2\}$.

- $C \leftarrow \mathsf{KZG.Com}(\mathsf{crs}, f)$: Suppose $f(\boldsymbol{x}) = \sum_{\boldsymbol{b} \in B_\mu} f_{\boldsymbol{b}} \cdot \mathsf{eq}(\boldsymbol{b}, \boldsymbol{x})$. Computes $C := \sum_{\boldsymbol{b} \in B_\mu} f_{\boldsymbol{b}} \cdot [\mathsf{eq}(\boldsymbol{b}, \boldsymbol{\tau})]_1$.
- $\boldsymbol{\pi} \leftarrow \mathsf{KZG.Open}(f, \boldsymbol{a}, \mathsf{crs})$: To prove $y := f(\boldsymbol{a})$ on point $\boldsymbol{a}$. Computes the multilinear polynomials $q_i(\boldsymbol{x})$ such that $f(\boldsymbol{x}) - y = \sum_{i \in [\mu]} (x_i - a_i) q_i(\boldsymbol{x})$. Then computes $\pi_i = [q_i(\boldsymbol{\tau})]_1$, and takes $\boldsymbol{\pi} = (\pi_1, ..., \pi_\mu)$ as proof of the opening.
- $b \leftarrow \mathsf{KZG.Verify}(C, \boldsymbol{\pi}, \boldsymbol{a}, y, \mathsf{crs})$: Check if

$$e(C - [y]_1, [1]_2) = \prod_{i \in [\mu]} e(\pi_i, [\tau_i - a_i]_2).$$

  Returns 1 if the equality holds, else 0.

4 PIOP for New Constraint System

In this section, we first introduce a PIOP for proving the linear constraints. Then, we describe the complete PIOP for proving the new constraint system.

4.1 PIOP for linear constraint

We begin by formalizing the linear constraint relation $\mathcal{R}_{\mathsf{linear}}$. To keep the definition general, we consider a witness polynomial T over ν variables.

Definition 5 (Linear constraint relation). *Given $w \in \mathbb{N}$, the relation $\mathcal{R}_{\mathsf{linear}}$ is the set of all tuples*

$$(\mathbb{i}; \mathbb{x}; \mathbb{w}) = (\mathcal{Q}; [[T]]; T),$$

where $\mathcal{Q} := \left(\mathcal{Q}^{(0)}, \ldots, \mathcal{Q}^{(w-1)}\right)$ is a set of linear constraints, each $\mathcal{Q}^{(t)} := (\beta_j^{(t)}, \boldsymbol{b}_j^{(t)})_{j=0}^{|\mathcal{Q}^{(t)}|-1}$ with coefficients $\beta_j^{(t)} \in \mathbb{F}$ and indices $\boldsymbol{b}_j^{(t)} \in B_\nu$. $T \in \mathbb{F}_\nu^{\leq 1}[\boldsymbol{X}]$ is a multilinear polynomial such that for all $t \in [0, w-1]$, $\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot T(\boldsymbol{b}_i^{(t)}) = 0$.

We combine these w constraints into a single sumcheck instance using a selector polynomial $W_r \in \mathbb{F}_\nu^{\leq 1}[\boldsymbol{X}]$. Let $N = 2^\nu$, W_r is constructed through Algorithm 1, where $\mathbf{w}[\boldsymbol{b}]$ denotes the entry of $\mathbf{w}$ indexed by the integer whose binary representation is $\boldsymbol{b}$.

Our PIOP protocol Π_{linear} for the relation $\mathcal{R}_{\mathsf{linear}}$ is presented in Fig. 7. In the offline phase, the indexer outputs an oracle $[[W]]$ to the polynomial W. While in the online phase, the verifier uses this oracle to check the consistency between the random point evaluation of W_r and W through the quotient polynomial verification.

Theorem 4. *The PIOP Π_{linear} is perfectly complete, and has a knowledge soundness error of at most $\frac{3\nu+2w}{|\mathbb{F}|}$.*

The proof is presented in Appendix A.2.

Algorithm 1 Construct selector polynomial for $\mathcal{R}_{\mathsf{linear}}$

Algorithm $\mathsf{SelectPoly}(\mathcal{Q}, r, N)$

$\mathbf{w} \leftarrow [0; N]$ ▷ Initialize vector with zeros
for $t = 0$ to $w - 1$ **do**
  for $i = 0$ to $|\mathcal{Q}^{(t)}| - 1$ **do**
    $\mathbf{w}[\boldsymbol{b}_i^{(t)}] \leftarrow \mathbf{w}[\boldsymbol{b}_i^{(t)}] + \beta_i^{(t)} \cdot r^t$
  end for
end for
Define W_r such that $W_r(\boldsymbol{b}) = \mathbf{w}[\boldsymbol{b}]$ for all $\boldsymbol{b} \in B_\nu$
return W_r

Offline phase: $\mathcal{I}(\mathbb{i})$ runs

- $\mathcal{I}$ constructs the polynomial $W(\boldsymbol{X}, Y) := \sum_{t=0}^{w-1} Y^t \big(\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot \mathsf{eq}(\boldsymbol{b}_i^{(t)}, \boldsymbol{X}) \big)$.
- $\mathcal{I}$ outputs the oracle $[[W]]$.

Online phase: $\mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ and $\mathcal{V}(\mathbb{x}, [[W]])$ execute:

1. $\mathcal{V}$ samples randomness $r \xleftarrow{\$} \mathbb{F}$ and sends r to $\mathcal{P}$.
2. $\mathcal{P}$ computes:
   - $W_r \leftarrow \mathsf{SelectPoly}(\mathcal{Q}, r, N)$ using Algorithm 1,
   - $Q(\boldsymbol{X}, Y) := \frac{W(\boldsymbol{X}, Y) - W_r(\boldsymbol{X})}{Y - r}$, which is multilinear in $\boldsymbol{X}$ and of degree at most $w - 2$ in Y.

   $\mathcal{P}$ sends oracles $[[W_r]], [[Q]]$ to $\mathcal{V}$.
3. $\mathcal{V}$ samples $\boldsymbol{\alpha} \in \mathbb{F}^\nu, \gamma \in \mathbb{F}$ and queries $Q(\boldsymbol{\alpha}, \gamma)$, $W(\boldsymbol{\alpha}, \gamma)$, $W_r(\boldsymbol{\alpha})$.
4. $\mathcal{V}$ checks $Q(\boldsymbol{\alpha}, \gamma)(\gamma - r) = W(\boldsymbol{\alpha}, \gamma) - W_r(\boldsymbol{\alpha})$.
5. Let $F(\boldsymbol{X}) := W_r(\boldsymbol{X}) \cdot T(\boldsymbol{X})$. $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck PIOP Π_{sum} for $(0, [[F]]; F) \in \mathcal{R}_{\mathsf{sum}}$.

Fig. 7. PIOP Π_{linear} for $\mathcal{R}_{\mathsf{linear}}$.

4.2 Pancake PIOP construction

Let the public input be $\mathsf{x} \in \mathbb{F}^\ell$. Let k_{w} denote the number of actual input and output wires per gate, let $j_{\mathsf{out}} \in [0, k_{\mathsf{w}} - 1]$ denote the output-wire position, and let $\ell_{\mathsf{w}} = 2^{\nu_{\mathsf{w}}}$ where $\nu_{\mathsf{w}} := \lceil \log_2 k_{\mathsf{w}} \rceil$ be the padded encoding width. Let $N = 2^\nu$ be the padded number of non-addition-gate and public-input rows.

The witness polynomial $T \in \mathbb{F}^{\leq 1}_{\nu_{\mathsf{w}}+\nu}[\boldsymbol{Z}, \boldsymbol{X}]$ is multilinear in $(\nu_{\mathsf{w}} + \nu)$ variables. It encodes the public inputs in the output wire. Specifically, we require that for $j \in [0, \ell - 1]$,

$$T(\langle j_{\mathsf{out}} \rangle_{\nu_{\mathsf{w}}}, \langle j \rangle_\nu) = \mathsf{x}_j.$$

The slots indexed by $j \in [k_{\mathsf{w}}, \ell_{\mathsf{w}} - 1]$ are padding coordinates used only to obtain a Boolean-hypercube encoding. The honest encoder sets them to zero, while the relation and the verifier depend only on the first k_{w} slices.

To verify this encoding, the verifier constructs a public input polynomial $\mathsf{PI} \in \mathbb{F}_{\nu}^{\leq 1}[\boldsymbol{X}]$, where $\mathsf{PI}(\langle i \rangle_{\nu}) = \mathsf{x}_i$ for $i \in \{0, \ldots, \ell - 1\}$, and $\mathsf{PI}(\boldsymbol{x}) = 0$ for all other $\boldsymbol{x} \in B_{\nu}$. The verifier uses PI to check that the public inputs are correctly embedded in the witness polynomial.

Definition 6 (Pancake relation). *Fix public parameters*

$$\mathsf{gp} := (\mathbb{F}, \ell, N, k_{\mathsf{w}}, \ell_{\mathsf{w}}, j_{\mathsf{out}}, \ell_{\mathsf{q}}, f, w),$$

where:

- ℓ *is the public input length.*
- $N = 2^{\nu}$ *is the number of rows after padding the non-addition-gate and public-input rows with dummy rows.*
- k_{w} *is the number of actual input and output wires per gate,* $j_{\mathsf{out}} \in [0, k_{\mathsf{w}} - 1]$ *is the output-wire position, and* $\ell_{\mathsf{w}} = 2^{\nu_{\mathsf{w}}}$ *with* $\nu_{\mathsf{w}} = \lceil \log_2 k_{\mathsf{w}} \rceil$ *is the padded encoding width.*
- $\ell_{\mathsf{q}} \in \mathbb{N}$ *is the number of gate selector polynomials.*
- $f : \mathbb{F}^{1+\ell_{\mathsf{q}}+k_{\mathsf{w}}} \to \mathbb{F}$ *is an algebraic map of degree* d*.*
- w*: total number of linear constraints.*

The relation $\mathcal{R}_{\mathsf{pcake}}$ *consists of tuples*

$$(\mathbb{i}; \mathbb{x}; \mathbb{w}) = (\mathcal{Q}, \{S_i\}_{i=0}^{\ell_{\mathsf{q}}-1}; \mathsf{PI}; T),$$

where:

- $\mathcal{Q} := \left(\mathcal{Q}^{(0)}, \ldots, \mathcal{Q}^{(w-1)}\right)$ *is a linear constraint set. For each* $t \in [0, w-1]$*, we define* $\mathcal{Q}^{(t)} := (\beta_j^{(t)}, \boldsymbol{b}_j^{(t)})_{j=0}^{|\mathcal{Q}^{(t)}|-1}$*, where* $\beta_j^{(t)} \in \mathbb{F}$ *and* $\boldsymbol{b}_j^{(t)} \in B_{\nu_{\mathsf{w}}+\nu}$*. Writing* $\boldsymbol{b}_j^{(t)} = (\boldsymbol{z}_j^{(t)}, \boldsymbol{x}_j^{(t)})$*, every referenced wire index satisfies* $\boldsymbol{z}_j^{(t)} = \langle h \rangle_{\nu_{\mathsf{w}}}$ *for some* $h \in [0, k_{\mathsf{w}} - 1]$*.*
- $S_0, \ldots, S_{\ell_{\mathsf{q}}-1} \in \mathbb{F}_{\nu}^{\leq 1}[\boldsymbol{X}]$ *are gate selector polynomials,*
- $\mathsf{PI} \in \mathbb{F}_{\nu}^{\leq 1}[\boldsymbol{X}]$ *is public input polynomial,*
- $T \in \mathbb{F}_{\nu_{\mathsf{w}}+\nu}^{\leq 1}[\boldsymbol{Z}, \boldsymbol{X}]$ *is witness polynomial,*

such that

- ***Gate constraints:*** $\forall \boldsymbol{x} \in B_{\nu}$*,*

$$0 = f\left(\mathsf{PI}(\boldsymbol{x}), S_0(\boldsymbol{x}), \ldots, S_{\ell_{\mathsf{q}}-1}(\boldsymbol{x}), T(\langle 0 \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{x}), \ldots, T(\langle k_{\mathsf{w}} - 1 \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{x})\right).$$

  Here, the function f *encodes the arithmetic relations of the circuit's non-addition gates (e.g., multiplication and custom gates).*
- ***Linear constraints:***

$$\forall t \in [0, w-1] : \textstyle\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot T(\boldsymbol{b}_i^{(t)}) = 0.$$

Offline phase: $\mathcal{I}(\mathtt{i})$ runs

- $\mathcal{I}$ calls the PIOP Π_{linear} indexer $[[W]] \leftarrow \mathcal{I}_{\mathsf{linear}}(\mathcal{Q})$.
- $\mathcal{I}$ outputs oracles $\mathcal{O} = \{[[W]], [[S_0]], \ldots, [[S_{\ell_{\mathsf{q}}-1}]]\}$.

Online phase: $\mathcal{P}(\mathsf{gp}, \mathtt{i}, \mathtt{x}, \mathtt{w})$ and $\mathcal{V}(\mathsf{gp}, \mathtt{x}, \mathcal{O})$ run

1. $\mathcal{P}$ sends the witness oracle $[[T]]$ to $\mathcal{V}$.
2. **Gate Check:** The prover runs the ZeroCheck PIOP Π_{zero} (Section 3.4) to show $([[\mathsf{G}]]; \mathsf{G}) \in \mathcal{R}_{\mathrm{zero}}$, where the virtual polynomial G is defined as:
$$\mathsf{G}(\boldsymbol{X}) := f\begin{pmatrix}\mathsf{PI}(\boldsymbol{X}), S_0(\boldsymbol{X}), \ldots, S_{\ell_{\mathsf{q}}-1}(\boldsymbol{X}),\\ T(\langle 0\rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X}), \ldots, T(\langle k_{\mathsf{w}}-1\rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X})\end{pmatrix}.$$
3. **Linear Check:** The prover runs the PIOP Π_{linear} to show that
$$(\mathcal{Q}; [[T]]; T) \in \mathcal{R}_{\mathsf{linear}}.$$

Fig. 8. PIOP Π_{pcake} for $\mathcal{R}_{\mathsf{pcake}}$.

For the complexity bounds only, we assume that the total number of terms across all linear constraints satisfies
$$\sum_{t=0}^{w-1} |\mathcal{Q}^{(t)}| = O(N + w).$$

Now, we present our PIOP Π_{pcake} for $\mathcal{R}_{\mathsf{pcake}}$.

Theorem 5. *Let* gp *be the public parameters from Definition 6, where* $k_{\mathsf{w}}, \ell_{\mathsf{w}}, \ell_{\mathsf{q}} = O(1)$*, and let* $d = \deg(f)$*. The PIOP* Π_{pcake} *in Fig. 8 is perfectly complete and has knowledge-soundness error* $O((d\nu + w)/|\mathbb{F}|)$*. Its query complexity is* $2\nu + \nu_{\mathsf{w}} + 4$*, its round complexity is* $2\nu + \nu_{\mathsf{w}} + 3$*, its prover time and proof-oracle size are* $O(wN)$*, and its verifier time is* $O(\nu)$*.*

The proof is presented in Appendix A.3.

5 Pancake: linear-time prover SNARK with linear constraints

In this section, we introduce Pancake, our new SNARK construction that achieves linear-time prover efficiency. The adoption of our new constraint system reduces the size of the witness polynomial, resulting in substantial improvements in the prover's efficiency.

Our construction leverages the multilinear KZG polynomial commitment [36], as elaborated in Section 3.5, together with a univariate KZG key for the committed sumcheck. The trusted setup samples independent field elements

$(\boldsymbol{\tau}, \tau_y, s)$. The pair $(\boldsymbol{\tau}, \tau_y)$ enables the commitment to a $(\nu+1)$-variate polynomial $f(\boldsymbol{X}, Y)$ as $[f(\boldsymbol{\tau}, \tau_y)]_1$, while s is used only by the univariate sub-scheme.

Recall that a key step in Pancake is verifying the polynomial identity in Equation (4). The KZG PCS enables efficient verification via the pairing check:

$$e([Q(\boldsymbol{\tau}, \tau_y)]_1, [\tau_y - r]_2) = e([W(\boldsymbol{\tau}, \tau_y)]_1 - [W_r(\boldsymbol{\tau})]_1, [1]_2).$$

To maintain linear-time complexity, the prover avoids explicitly constructing and committing to Q in the online phase. Instead, Pancake leverages the decomposition established in Equation (5), where $Q(\boldsymbol{X}, Y)$ is expressed as a linear combination $\sum_{i=0}^{w-2} r^i \cdot p_i(\boldsymbol{X}, Y)$. Since the commitments C_i to p_i are precomputed during the Setup phase, the prover simply assembles the commitment to Q as $C_Q = \sum_{i=0}^{w-2} r^i \cdot C_i$ in the online phase. This operation requires only $O(w)$ group operations, where w is the number of linear constraints.

5.1 Our construction

Let $T \in \mathbb{F}^{\leq 1}_{\nu_{\mathsf{w}}+\nu}[\boldsymbol{Z}, \boldsymbol{X}]$ be the multilinear witness polynomial. For batched PCS openings, we decompose T into $\ell_{\mathsf{w}} = 2^{\nu_{\mathsf{w}}}$ polynomials $\{T_j\}_{j=0}^{\ell_{\mathsf{w}}-1}$ defined by

$$\forall j \in [0, \ell_{\mathsf{w}} - 1],\ T_j(\boldsymbol{X}) := T(\langle j \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X}) \in \mathbb{F}^{\leq 1}_{\nu}[\boldsymbol{X}].$$

For the canonical honest encoding, we set $T_j \equiv 0$ at the padded indices $j \in [k_{\mathsf{w}}, \ell_{\mathsf{w}} - 1]$. Neither the gate relation nor any linear constraint references a padded slot. Consequently, the online protocol uses only the first k_{w} slices. The polynomials W_j, $W_{r,j}$, and $H_t(\langle j \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X})$ are identically zero at padded indices, and their commitments are omitted. Following the same principle, we split the polynomial W into ℓ_{w} partial $\nu+1$-variate polynomials W_j, such that $W_j(\boldsymbol{X}, Y) = W(\langle j \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X}, Y)$. We then have

$$W_j(\boldsymbol{X}, Y) := \textstyle\sum_{t=0}^{w-1} Y^t \cdot H_t(\langle j \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X}), \tag{6}$$

which is multilinear in $\boldsymbol{X}$ and of degree at most $w-1$ in Y, where the term H_t is defined as:

$$H_t(\boldsymbol{Z}, \boldsymbol{X}) := \textstyle\sum_{i=0}^{|\mathcal{Q}^{(t)}|-1} \beta_i^{(t)} \cdot \mathsf{eq}(\boldsymbol{b}_i^{(t)}, (\boldsymbol{Z}, \boldsymbol{X})) \tag{7}$$

Recall the selector polynomial $W_r \in \mathbb{F}^{\leq 1}_{\nu_{\mathsf{w}}+\nu}[\boldsymbol{Z}, \boldsymbol{X}]$ is obtained by running Algorithm 1. To construct the corresponding partial polynomials $W_{r,j} \in \mathbb{F}^{\leq 1}_{\nu}[\boldsymbol{X}]$ such that $W_{r,j}(\boldsymbol{X}) = W_r(\langle j \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X})$, we present Algorithm 2 as:

Linear Offline Setup Phase Our protocol relies on a circuit-specific trusted setup. While a universal setup is possible, naively committing to the polynomials W_j and the basis polynomials $p_i^{(j)}$ (of degree w in Y) would require $O(w \cdot N)$ time. To achieve linear time, we exploit a recurrence relation satisfied by the quotient basis polynomials $p_i^{(j)}$:

$$p_{i-1}^{(j)}(\boldsymbol{X}, Y) = Y \cdot p_i^{(j)}(\boldsymbol{X}, Y) + H_i(\langle j \rangle_{\nu_{\mathsf{w}}}, \boldsymbol{X}).$$

Algorithm 2 Construct partial selector polynomials

Algorithm $\mathsf{SelectPolys}(\mathcal{Q}, r, \ell_{\mathsf{w}}, N)$

$\mathbf{w} \leftarrow [0; \ell_{\mathsf{w}} \cdot N]$
for $t = 0$ to $w - 1$ **do**
  for $i = 0$ to $|\mathcal{Q}^{(t)}| - 1$ **do**
    $\mathbf{w}[\boldsymbol{b}_i^{(t)}] \leftarrow \mathbf{w}[\boldsymbol{b}_i^{(t)}] + \beta_i^{(t)} \cdot r^t$
  end for
end for
Define $\{W_{r,j}\}_{j=0}^{\ell_{\mathsf{w}}-1}$ such that $W_{r,j}(\langle k \rangle_\nu) = \mathbf{w}[k \cdot \ell_{\mathsf{w}} + j]$ for $k \in [0, N-1]$, since the point $(\langle j \rangle_{\nu_{\mathsf{w}}}, \langle k \rangle_\nu)$ has integer index $j + \ell_{\mathsf{w}} \cdot k$ under the little-endian convention for $\langle \cdot \rangle_m$
return $W_{r,0}, \ldots, W_{r,\ell_{\mathsf{w}}-1}$

We present the Setup algorithm in Algorithm 3. By precomputing a table for eq evaluations and leveraging the above recurrence, it generates all necessary commitments using $O(N + w + \sum_t |\mathcal{Q}^{(t)}| + d)$ group operations. Under this bounded-incidence assumption, and for $d = O(1)$, the setup and online prover costs are $O_\lambda(N + w)$. The scalar table M and the trapdoors $(\boldsymbol{\tau}, \tau_y, s)$ are used only within the trusted setup and are destroyed once Setup outputs.

Online Protocol We now present the online phase of $\mathsf{Pancake}$. $\mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w})$ and $\mathcal{V}(\mathsf{vp}, \mathbb{x})$ execute the following:

1. $\mathcal{P}$ computes witness commitments $\hat{T}_j := [T_j(\boldsymbol{\tau})]_1$ for all $j \in [0, k_{\mathsf{w}} - 1]$ and sends them to $\mathcal{V}$.
2. $\mathcal{V}$ samples and sends the challenge $r \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
3. $\mathcal{P}$ computes $\{W_{r,j}\}_{j=0}^{\ell_{\mathsf{w}}-1} \leftarrow \mathsf{SelectPolys}(\mathcal{Q}, r, \ell_{\mathsf{w}}, N)$. For each $j \in [0, k_{\mathsf{w}} - 1]$, $\mathcal{P}$ computes the commitment to $W_{r,j}$ as
$$\hat{C}_{W,j} := \textstyle\sum_{t=0}^{w-1} r^t \cdot \hat{H}_{j,t}.$$
and sends $\hat{C}_{W,j}$ to $\mathcal{V}$. Note that $W_{r,j}(\boldsymbol{X})$ is the specialization $W_j(\boldsymbol{X}, r)$ as in Equation (6).
4. $\mathcal{V}$ samples and sends the challenge $\gamma \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
5. $\mathcal{P}$ computes the batched quotient commitment:
$$\hat{C}_{Q,j} := \textstyle\sum_{i=0}^{w-2} r^i \cdot \hat{C}_{j,i}, \quad \hat{C}_Q := \sum_{j=0}^{k_{\mathsf{w}}-1} \gamma^j \cdot \hat{C}_{Q,j}.$$
$\mathcal{P}$ sends $\hat{C}_Q$ to $\mathcal{V}$.
6. For each $j \in [0, k_{\mathsf{w}} - 1]$, $\mathcal{V}$ computes $\hat{E}_j := \hat{W}_j - \hat{C}_{W,j}$, and checks
$$e(\hat{C}_Q, [\tau_y - r]_2) \stackrel{?}{=} e\left(\textstyle\sum_{j=0}^{k_{\mathsf{w}}-1} \gamma^j \cdot \hat{E}_j, [1]_2\right).$$
7. $\mathcal{V}$ samples and sends the challenges $\alpha \xleftarrow{\$} \mathbb{F}$, $\boldsymbol{z} \xleftarrow{\$} \mathbb{F}^\nu$ to $\mathcal{P}$.

Algorithm 3 Linear-Time Circuit-Specific Trusted Setup

Algorithm $\mathsf{Setup}(\mathsf{gp}, \mathcal{Q}, \{S_i\}_{i=0}^{\ell_\mathsf{q}-1})$

Sample $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_\nu) \xleftarrow{\$} \mathbb{F}^\nu, \tau_y \xleftarrow{\$} \mathbb{F}$, and an independent $s \xleftarrow{\$} \mathbb{F}$ for the univariate sub-scheme.
Compute $[\tau_i]_2 \in \mathbb{G}_2$ for $i \in [\nu]$.
Set $\mathsf{vk}_\mathsf{ml} := ([1]_2, \{[\tau_i]_2\}_{i\in[\nu]})$, $\mathsf{vk}_\mathsf{quo} := ([1]_2, [\tau_y]_2)$, and $\mathsf{vk}_\mathsf{uni} := ([1]_2, [s]_2)$.
Set $\mathsf{srs}_\mathsf{v} := (\mathsf{vk}_\mathsf{ml}, \mathsf{vk}_\mathsf{quo}, \mathsf{vk}_\mathsf{uni})$.
— **Precompute eq table** M —
Construct table M**:** Compute and store $\mathsf{M}[\boldsymbol{b}] := \mathsf{eq}(\boldsymbol{b}, \boldsymbol{\tau})$ for all $\boldsymbol{b} \in B_\nu$ in $O(N)$ time [47].
Set $\mathsf{ck}_\mathsf{ml} := \{[\mathsf{eq}(\boldsymbol{b}, \boldsymbol{\tau})]_1\}_{\boldsymbol{b}\in B_\nu}$ and $\mathsf{ck}_\mathsf{uni} := \{[s^k]_1\}_{k=0}^{d-1}$.
Set $\mathsf{srs}_\mathsf{p} := (\mathsf{srs}_\mathsf{v}, \mathsf{ck}_\mathsf{ml}, \mathsf{ck}_\mathsf{uni})$.
— **Commit to selector and constraint polynomials** —
Compute $\hat{S}_i \leftarrow [S_i(\boldsymbol{\tau})]_1$ for $i \in [0, \ell_\mathsf{q} - 1]$ using M.
Compute $\hat{W}_j \leftarrow [W_j(\boldsymbol{\tau}, \tau_y)]_1$ for $j \in [0, k_\mathsf{w} - 1]$ using M.
— **Compute quotient basis commitments** —
for $j = 0$ to $k_\mathsf{w} - 1$ **do**
    For $t = 0, \ldots, w - 1$, compute:
        $\hat{H}_{j,t} \leftarrow [H_t(\langle j \rangle_{\nu_\mathsf{w}}, \boldsymbol{\tau})]_1$ using M.
    Base case for $p_i^{(j)}$**:** Set $\hat{C}_{j,w-2} \leftarrow \hat{H}_{j,w-1}$.
    Recursive step: For $i = w - 2$ down to 1:
        $\hat{C}_{j,i-1} \leftarrow \tau_y \cdot \hat{C}_{j,i} + \hat{H}_{j,i}$.
end for
— **Output** —
return $\mathsf{vp} := \left(\mathsf{srs}_\mathsf{v}, \{\hat{S}_i\}_{i=0}^{\ell_\mathsf{q}-1}, \{\hat{W}_j\}_{j=0}^{k_\mathsf{w}-1}\right)$,

$$\mathsf{pp} := \left(\mathsf{srs}_\mathsf{p}, \{\hat{C}_{j,i}, \hat{H}_{j,t}\}_{\substack{j\in[0,k_\mathsf{w}-1],\\ i\in[0,w-2]\\ t\in[0,w-1]}}, \{\hat{S}_i\}_{i=0}^{\ell_\mathsf{q}-1}\right).$$

8. Let polynomial $\mathsf{W} \in \mathbb{F}_\nu^{\leq 2}[\boldsymbol{X}], \mathsf{G} \in \mathbb{F}_\nu^{\leq d}[\boldsymbol{X}]$,

$$\mathsf{W} := W_{r,0} \cdot T_0 + \cdots + W_{r,k_\mathsf{w}-1} \cdot T_{k_\mathsf{w}-1}.$$
$$\mathsf{G} := f\left(\mathsf{PI}, S_0, \ldots, S_{\ell_\mathsf{q}-1}, T_0, \ldots, T_{k_\mathsf{w}-1}\right).$$

   $\mathcal{P}$ and $\mathcal{V}$ run the sumcheck to prove

$$\textstyle\sum_{\boldsymbol{b}\in B_\nu} \left(\mathsf{W}(\boldsymbol{b}) + \alpha \cdot (\mathsf{G}(\boldsymbol{b}) \cdot \mathsf{eq}(\boldsymbol{b}, \boldsymbol{z}))\right) = 0,$$

   where the sumcheck is of size ν, and has degree $d + 1$.
9. The sumcheck protocol proceeds in several rounds. In each round, $\mathcal{P}$ makes claims about certain polynomial evaluations. These evaluation claims are proved via the univariate batch-opening algorithm of Section C.2 in [6], producing a single $\mathbb{G}_1$ proof element.
10. In the final round of the sumcheck protocol, $\mathcal{V}$ queries the evaluations $\{T_j(\boldsymbol{a}), W_{r,j}(\boldsymbol{a})\}_{j=0}^{k_\mathsf{w}-1}$, $\{S_i(\boldsymbol{a})\}_{i=0}^{\ell_\mathsf{q}-1}$ and computes $\mathsf{PI}(\boldsymbol{a})$.
11. $\mathcal{V}$ samples random challenges $\zeta_0, \ldots, \zeta_{\ell_\mathsf{q}+2k_\mathsf{w}-1}$ and sends them to $\mathcal{P}$.

12. $\mathcal{P}$ computes the batched polynomial B and its opening proof
$$B := \sum_{i=0}^{\ell_{\mathsf{q}}-1} \zeta_i \cdot S_i + \sum_{j=0}^{k_{\mathsf{w}}-1} \zeta_{j+\ell_{\mathsf{q}}} \cdot T_j + \sum_{j=0}^{k_{\mathsf{w}}-1} \zeta_{j+\ell_{\mathsf{q}}+k_{\mathsf{w}}} \cdot W_{r,j}.$$
$\boldsymbol{\pi}_o \leftarrow \mathsf{KZG.Open}(B, \boldsymbol{a}, \mathsf{srs_p})$. $\mathcal{P}$ sends $\boldsymbol{\pi}_o$ to $\mathcal{V}$.
13. $\mathcal{V}$ computes the expected commitment to B and the expected evaluation v_B:
$$C_B := \sum_{i=0}^{\ell_{\mathsf{q}}-1} \zeta_i \cdot \hat{S}_i + \sum_{j=0}^{k_{\mathsf{w}}-1} \zeta_{j+\ell_{\mathsf{q}}} \cdot \hat{T}_j + \sum_{j=0}^{k_{\mathsf{w}}-1} \zeta_{j+\ell_{\mathsf{q}}+k_{\mathsf{w}}} \cdot \hat{C}_{W,j},$$
$$v_B := \sum_{i=0}^{\ell_{\mathsf{q}}-1} \zeta_i \cdot S_i(\boldsymbol{a}) + \sum_{j=0}^{k_{\mathsf{w}}-1} \zeta_{j+\ell_{\mathsf{q}}} \cdot T_j(\boldsymbol{a}) + \sum_{j=0}^{k_{\mathsf{w}}-1} \zeta_{j+\ell_{\mathsf{q}}+k_{\mathsf{w}}} \cdot W_{r,j}(\boldsymbol{a}).$$
14. $\mathcal{V}$ checks $\mathsf{KZG.Verify}(C_B, \boldsymbol{\pi}_o, \boldsymbol{a}, v_B, \mathsf{srs_v})$ and accepts if it returns 1.

Theorem 6. *Let* $D_{\mathsf{aug}} := \nu + w + d + 2$ *and* $\epsilon_{\mathsf{lift}} := \mathsf{Adv}^{(D_{\mathsf{aug}}, D_{\mathsf{aug}})\text{-}\mathsf{DLOG}}(\lambda) + \frac{D_{\mathsf{aug}}}{|\mathbb{F}|}$. *Assume the* $(D_{\mathsf{aug}}, D_{\mathsf{aug}})$*-DLOG assumption and that the multilinear and univariate KZG commitments satisfy witness-extended emulation with negligible error* ϵ_{wee}. *In the Algebraic Group Model, the interactive* Pancake *protocol is a perfectly complete argument of knowledge for* $\mathcal{R}_{\mathsf{pcake}}$. *Its knowledge error is at most*
$$\epsilon_{\mathsf{lift}} + \epsilon_{\mathsf{wee}} + \frac{w + \ell_{\mathsf{w}} + 3\nu}{|\mathbb{F}|} + \frac{\nu(d + 1 + k_{\mathsf{w}})}{|\mathbb{F}| - 2},$$
and hence is $O(((d + k_{\mathsf{w}} + 4)\nu + w + \ell_{\mathsf{w}} + D_{\mathsf{aug}})/|\mathbb{F}|) + \mathsf{negl}(\lambda)$.

The proof is presented in Appendix A.5.

Fiat–Shamir transcript The non-interactive protocol derives every verifier challenge from a domain-separated hash of the transcript prefix. The initial transcript contains a protocol identifier, gp, a digest of the circuit index and the public parameters $(\mathsf{pp}, \mathsf{vp})$, and the public input $\mathbb{x}$. The prover and verifier then apply the following order:

1. absorb $\{\hat{T}_j\}_{j=0}^{k_{\mathsf{w}}-1}$ and derive r;
2. absorb $\{\hat{C}_{W,j}\}_{j=0}^{k_{\mathsf{w}}-1}$ and derive γ;
3. absorb $\hat{C}_Q$ and derive $(\alpha, \boldsymbol{z})$;
4. execute the sumcheck protocol, proving its evaluation claims via the univariate batch-opening algorithm of Section C.2 in [6]; every verifier challenge is derived from a domain-separated hash of the corresponding transcript prefix;
5. after the final round, absorb all queried evaluations and the completed univariate batch-opening transcript;
6. derive $\zeta_0, \ldots, \zeta_{\ell_{\mathsf{q}}+2k_{\mathsf{w}}-1}$, absorb the multilinear opening proof $\boldsymbol{\pi}_o$, and output the proof.

All field challenges are obtained by an unbiased hash-to-field procedure and range over all of $\mathbb{F}$, as in the interactive protocol.

Corollary 1. *Under the assumptions of Theorem 6 and the security of the Fiat–Shamir transform for the resulting multi-round public-coin protocol [1,49], the transcript described above yields a succinct non-interactive argument of knowledge for* $\mathcal{R}_{\mathsf{pcake}}$ *in the Algebraic Group Model and the Random Oracle Model.*

The proof is presented in Appendix A.6.

5.2 Complexity

- **Prover time**: $O_\lambda(N + w)$.
- **Verifier time**: $O_\lambda(\nu)$.
- **Proof size**: Total $2\nu + 2k_{\mathsf{w}} + \ell_{\mathsf{q}}$ $\mathbb{F}$ elements and $2\nu + 2k_{\mathsf{w}} + 2$ $\mathbb{G}_1$ elements. Padded witness slots contribute no proof elements.
  - $\mathbb{F}$ elements: 2ν from sumcheck rounds, and $(2k_{\mathsf{w}}+\ell_{\mathsf{q}})$ from the evaluations of $T_j, S_j, W_{r,j}$.
  - $\mathbb{G}_1$ **elements**: $2k_{\mathsf{w}} + 1$ from polynomial commitments $(\hat{T}_j, \hat{C}_{W,j}, \hat{C}_Q)$, ν from sumcheck rounds, 1 from the univariate batch-opening proof, and ν from the batched multilinear KZG evaluation proofs $\boldsymbol{\pi}_o$.

6 Implementation and Evaluation

Except for the parallel-scaling experiment in Fig. 1c, all benchmarks were conducted on a server with 64 GB of RAM and an Intel Core i9-13900KF CPU (24 cores and 32 threads). Our implementation is based on the HyperPlonk library[1], which utilizes the BLS12-381 curve to realize a multilinear PCS.

6.1 Proof-Size Comparison

The proof sizes of both HyperPlonk and Pancake are primarily determined by the circuit size and two circuit-dependent parameters: k_{w} (number of witness polynomials) and ℓ_{q} (number of selector polynomials, which grows with the number of supported gate types). We consider two common circuit configurations:

- Jellyfish Turbo Plonk circuit: $k_{\mathsf{w}} = 5$ (padded to $\ell_{\mathsf{w}} = 8$), $\ell_{\mathsf{q}} = 13$,
- Vanilla Plonk circuit: $k_{\mathsf{w}} = 3$ (padded to $\ell_{\mathsf{w}} = 4$), $\ell_{\mathsf{q}} = 5$.

For both systems, commitments and evaluations corresponding to publicly known zero-padding slices are omitted from the proof-size count. Let n denote the number of gates in the original circuit before the removal of addition gates, let $\mu = \lceil \log_2 n \rceil$, and let δ be the addition gate ratio. The HyperPlonk proof consists of:

- $3 + k_{\mathsf{w}} + 2\mu$ $\mathbb{G}_1$ elements,
- $4\mu + 9 + 2k_{\mathsf{w}} + \ell_{\mathsf{q}} + 2\lceil \log_2(8 + 2k_{\mathsf{w}} + \ell_{\mathsf{q}}) \rceil$ $\mathbb{F}$ elements.

Substituting these parameters yields the proof-size comparison shown in Fig. 9 (with $\delta = 0.5$). As illustrated, Pancake consistently produces smaller proofs.

[1] `https://github.com/EspressoSystems/HyperPlonk`

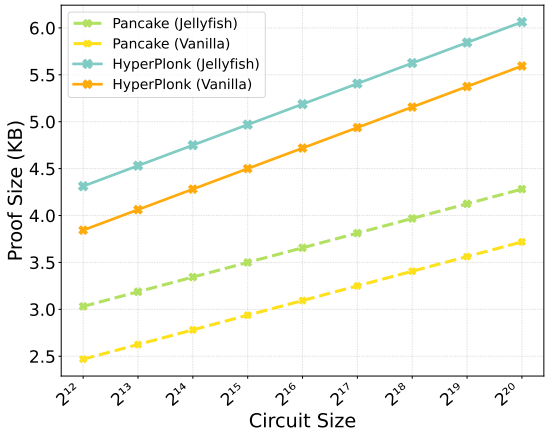


Fig. 9. Proof size comparison for $\delta = 0.5$, using compressed BLS12-381 encodings of 48 bytes per $\mathbb{G}_1$ element and 32 bytes per $\mathbb{F}$ element. Sizes are reported in decimal KB.

6.2 Performance Comparison

In this section, we evaluate the performance of Pancake against the baseline HyperPlonk [11]. We measure performance on both the Vanilla Plonk circuit [15] and the Jellyfish Turbo Plonk circuit[2]. Compared to the Vanilla circuit, the Jellyfish circuit supports high-degree custom gates (up to degree 5) and utilizes more witness polynomials, offering higher expressiveness with fewer gates for the same computation.

Survey of addition gate ratios in application circuits To contextualize our benchmarks, Table 2 lists the addition-gate ratio δ for several common circuits. We first compiled each benchmark program to R1CS using Circom [25], then converted it to a Plonkish circuit via Plonkify [53], and finally analyzed the gate distribution. The measured ratios range from 0.51 to 0.77, with most values lying between 0.5 and 0.7.

Table 2. Addition gate ratios δ for application circuits

Application	Jellyfish	Vanilla
Poseidon hash [21,42]	0.59	0.70
Curve point scalar multiplication [26]	0.56	0.73
SHA256 hash [27]	0.54	0.67
AES-encryption-128 [37]	0.65	0.77
Zk-rollup using Poseidon hash [43]	0.51	0.63

[2] `https://github.com/EspressoSystems/jellyfish/tree/HyperPlonk-bench`

Verifier time The verifier time for Pancake is $O(\nu)$. To assess Pancake's verifier performance, we compare it with HyperPlonk using Jellyfish Turbo Plonk circuits at $\delta = 0.5$. As shown in Table 3, the verifier times of the two systems are comparable: Pancake is slightly slower for smaller circuits and slightly faster for larger ones.

Table 3. Verifier time in milliseconds on Jellyfish Turbo Plonk circuits with $\delta = 0.5$.

	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
HyperPlonk	9.4	9.7	10.3	10.6	11.4	11.5	11.8
Pancake	9.8	10.1	10.3	10.7	11.0	11.3	11.6

Comparison of prover time We benchmark prover time for Pancake and HyperPlonk in both single-threaded and multi-threaded settings. For Pancake, prover time depends on the addition-gate ratio δ. Hence, for each δ we (i) sample a random circuit with the target δ, (ii) derive the corresponding linear constraint sets, and (iii) measure the prover time. For HyperPlonk, prover time is insensitive to δ, so for each circuit size we report a single timing result.

Our prover time is dominated by multilinear extension operations and elliptic curve multi-exponentiations, both of which are highly parallelizable. Consequently, Pancake exhibits greater improvements over HyperPlonk as the level of parallelism increases.

Jellyfish Turbo Plonk circuit.

- **Single thread**: As shown in Table 4, Pancake consistently outperforms HyperPlonk, with the advantage being most pronounced at high addition-gate ratios. For the circuit size $n = 2^{24}$, Pancake achieves speedups of 2.37× ($\delta = 0.7$) and 1.67× ($\delta = 0.5$).
- **Multi-threaded (32 threads)**: As shown in Fig. 1a and Table 4, this advantage widens to 3.6× (δ=0.7) and 2.4× (δ=0.5) at $n = 2^{24}$.
- **Parallelism scaling**: Pancake demonstrates superior efficiency in leveraging parallel processing. The parallel-scaling results in Fig. 1c were obtained on a separate dual-socket server equipped with two AMD EPYC 7502 CPUs, each with 32 cores and 64 threads, for a total of 64 cores and 128 threads. On that hardware, for a 2^{22}-gate circuit ($\delta = 0.5$), Pancake achieves a 16.6× speedup when scaled from 1 to 128 threads, compared to 8.1× for HyperPlonk.

Vanilla Plonk circuit. We also evaluate performance on vanilla Plonk circuits, as shown in Fig. 1b and Table 4.

- **Single thread**: At $n = 2^{24}$, Pancake achieves a 2.2× ($\delta = 0.7$) and a 1.7× ($\delta = 0.5$) speedup.
- **Multi-threaded (32 threads)**: The speedups increase to 3.5× ($\delta = 0.7$) and 2.5× ($\delta = 0.5$).

Table 4. Prover time in seconds for Pancake at different addition-gate ratios δ, compared with HyperPlonk.

n	Single thread				32 threads			
	HyperPlonk	Pancake			HyperPlonk	Pancake		
		$\delta = 0.7$	$\delta = 0.5$	$\delta = 0.3$		$\delta = 0.7$	$\delta = 0.5$	$\delta = 0.3$
Jellyfish Turbo Plonk								
2^{14}	1.2	0.6	0.9	1.1	0.2	0.1	0.1	0.1
2^{15}	2.2	1.1	1.6	2.2	0.3	0.1	0.2	0.2
2^{16}	4.0	2.0	2.9	4.0	0.6	0.2	0.3	0.5
2^{17}	7.8	3.8	5.4	7.7	1.1	0.4	0.6	0.9
2^{18}	15.2	7.1	10.2	14.9	2.0	0.8	1.1	1.6
2^{19}	28.6	13.6	19.6	28.2	4.0	1.5	2.1	3.1
2^{20}	56.8	25.7	36.9	54.0	8.9	3.0	3.9	6.2
2^{21}	111.2	49.3	71.0	104.0	17.1	5.8	7.3	11.5
2^{22}	215.6	95.5	137.5	199.1	33.6	11.0	15.3	23.2
2^{23}	425.8	182.5	263.2	386.8	66.5	21.1	29.0	45.1
2^{24}	851.1	358.8	510.8	767.2	147.1	41.1	60.5	89.9
Vanilla Plonk								
2^{14}	0.7	0.4	0.5	0.7	0.2	0.1	0.1	0.1
2^{15}	1.4	0.7	1.0	1.4	0.2	0.1	0.1	0.2
2^{16}	2.5	1.3	1.8	2.5	0.4	0.2	0.2	0.3
2^{17}	4.8	2.4	3.4	4.8	0.8	0.3	0.4	0.5
2^{18}	9.3	4.4	6.4	9.3	1.5	0.5	0.7	1.1
2^{19}	17.5	8.4	12.2	17.4	3.0	1.0	1.3	2.1
2^{20}	35.3	15.8	23.0	33.3	6.2	1.9	2.6	4.0
2^{21}	68.4	30.2	43.7	63.9	12.3	3.6	4.9	8.0
2^{22}	132.3	59.2	84.0	121.6	24.1	7.0	10.0	15.2
2^{23}	261.7	117.0	159.5	235.1	47.4	13.6	19.2	29.9
2^{24}	518.6	230.6	307.9	465.2	94.6	26.8	38.0	60.0

References

1. Attema, T., Fehr, S., Klooß, M.: Fiat-shamir transformation of multi-round interactive proofs. In: Theory of Cryptography Conference. pp. 113–142. Springer (2022)
2. 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)
3. 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)
4. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 781–796 (2014)
5. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In:

Proceedings of the 3rd innovations in theoretical computer science conference. pp. 326–349 (2012)
6. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41. pp. 649–680. Springer (2021)
7. 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)
8. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 677–706. Springer (2020)
9. Campanelli, M., Fiore, D., Querol, A.: Legosnark: Modular design and composition of succinct zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2075–2092 (2019)
10. Campanelli, M., Gailly, N., Gennaro, R., Jovanovic, P., Mihali, M., Thaler, J.: Linear time prover snarks with constant size proofs and square root size universal setup. In: International Conference on Cryptology and Information Security in Latin America. pp. 331–351. Springer (2023)
11. 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)
12. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. 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. 738–768. Springer (2020)
13. Choudhuri, A.R., Garg, S., Goel, A., Sekar, S., Sinha, R.: Sublonk: Sublinear prover plonk. Proceedings on Privacy Enhancing Technologies (2024)
14. Dellepere, M., Mishra, P., Shirzad, A.: Garuda and pari: faster and smaller snarks via equifficient polynomial commitments. Cryptology ePrint Archive (2024)
15. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
16. Ganesh, C., Patranabis, S., Singh, N.: Samaritan: Linear-time prover snark from new multilinear polynomial commitments. Cryptology ePrint Archive (2025)
17. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32. pp. 626–645. Springer (2013)
18. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Proceedings of the forty-third annual ACM symposium on Theory of computing. pp. 99–108 (2011)
19. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali, pp. 203–225. ACM (2019)

20. 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)
21. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for {Zero-Knowledge} proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535 (2021)
22. Green, M., Hall-Andersen, M., Hennenfent, E., Kaptchuk, G., Perez, B., Van Laer, G.: Efficient proofs of software exploitability for real-world processors. Proceedings on Privacy Enhancing Technologies (2023)
23. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 305–326. Springer (2016)
24. Hopwood, D., Bowe, S., Taylor, J., Hornby, S.: The halo 2 zero-knowledge proving system. `https://zcash.github.io/halo2/` (2020), accessed: 2024-05-01
25. Iden3: Circom: Circuit compiler for zero-knowledge proofs. `https://github.com/iden3/circom` (2020), gitHub repository. [Online; accessed 2025-05-17]
26. iden3: circomlib: escalarmul_test.circom – test circuit for elliptic curve scalar multiplication. `https://github.com/iden3/circomlib/blob/master/test/circuits/escalarmul_test.circom` (2023)
27. iden3: circomlib: sha256/main.circom – sha256 circuit implementation. `https://github.com/iden3/circomlib/blob/master/circuits/sha256/main.circom` (2023), gitHub repository file. [Online; accessed 2025-05-17]
28. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1353–1370 (2018)
29. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 177–194. Springer (2010)
30. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732 (1992)
31. Kothapalli, A., Masserova, E., Parno, B.: Poppins: A direct construction for asymptotically optimal zksnarks. Cryptology ePrint Archive (2020)
32. Lee, J., Setty, S., Thaler, J., Wahby, R.: Linear-time and post-quantum zero-knowledge snarks for r1cs. Cryptology ePrint Archive (2021)
33. Lindell: Parallel coin-tossing and constant-round secure two-party computation. Journal of Cryptology **16**, 143–184 (2003)
34. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM (JACM) **39**(4), 859–868 (1992)
35. Miyamae, T., Kozakura, F., Nakamura, M., Morinaga, M.: Zgridbc: Zero-knowledge proof based scalable and privacy-enhanced blockchain platform for electricity tracking. IEICE TRANSACTIONS on Information and Systems **106**(7), 1219–1229 (2023)
36. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: Theory of Cryptography Conference. pp. 222–242. Springer (2013)
37. Pluto: Circom aes-gcm circuits. `https://github.com/pluto/aes-circuits` (2023), gitHub repository. [Online; accessed 2025-05-17]

38. Rathee, D., Policharla, G.V., Xie, T., Cottone, R., Song, D.: Zebra: Snark-based anonymous credentials for practical, private and accountable on-chain access control. Cryptology ePrint Archive (2022)
39. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)
40. Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: Proceedings of the 8th ACM European Conference on Computer Systems. pp. 71–84 (2013)
41. Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zksnarks. Cryptology ePrint Archive (2020)
42. Shivangi470: Poseidon circom circuits. `https://github.com/shivangi470/poseidon.circom` (2023)
43. stksax: zk_rollup: Circom-based zkrollup with eddsa verification and merkle tree batching. `https://github.com/stksax/zk_rollup` (2023), gitHub repository. [Online; accessed 2025-05-17]
44. Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-prime: a standard specification (sok). Cryptology ePrint Archive (2020)
45. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Annual Cryptology Conference. pp. 71–89. Springer (2013)
46. Thaler, J., et al.: Proofs, arguments, and zero-knowledge. Foundations and Trends® in Privacy and Security **4**(2–4), 117–660 (2022)
47. Vu, V., Setty, S., Blumberg, A.J., Walfish, M.: A hybrid architecture for interactive verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 223–237. IEEE (2013)
48. Wang, W., Papamanthou, C., Srinivasan, S., Papadopoulos, D.: Dynamic zk-snarks. Cryptology ePrint Archive (2024)
49. Wikström, D.: Special soundness in the random oracle model. Cryptology ePrint Archive (2021)
50. 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)
51. 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)
52. Xue, Y., Gao, P., Lu, X., Au, M.H.: PANCAKE: A SNARK with plonkish constraints, almost-free additions, no permutation check, and a linear-time prover. Cryptology ePrint Archive, Paper 2026/212 (2026), `https://eprint.iacr.org/2026/212`
53. Zhu, P.: Plonkify: R1cs-to-plonk transpiler. Cryptology ePrint Archive (2025)

A Proof of Theorems

A.1 Preliminary Lemmas

Lemma 2 (Schwartz-Zippel Lemma). *Let $f(x_1, \ldots, x_n) \neq 0$ be a nonzero polynomial with total degree d over a field $\mathbb{F}$, and let $S \subseteq \mathbb{F}$ be a finite set.*

If each x_i is sampled independently and uniformly at random from S, then $\Pr[f(x_1, \ldots, x_n) = 0] \leq d/|S|$.

Lemma 3. *[33] Any proof of knowledge for relation $\mathcal{R}$ also satisfies witness extended emulation for $\mathcal{R}$.*

Lemma 4 (Sound PIOPs are knowledge sound). *[11] If a PIOP for relation $\mathcal{R}$ has soundness error δ, and for all $(i; x; w) \in \mathcal{R}$, the witness w consists entirely of polynomials to which the verifier has oracle access, then the PIOP has knowledge soundness error δ, with an extractor running in time $O(|w|)$.*

A.2 Proof of Theorem 4

Proof. **Completeness.** Assume the witness T satisfies the linear constraints, i.e., $(\mathcal{Q}; [[T]]; T) \in \mathcal{R}_{\mathsf{linear}}$. According to Algorithm 1, the constructed polynomial W_r satisfies $W_r(\boldsymbol{x}) = W(\boldsymbol{x}, r)$ for all $\boldsymbol{x} \in B_\nu$. Therefore:

1. Since $W_r(\boldsymbol{X})$ and $W(\boldsymbol{X}, r)$ are both multilinear in $\boldsymbol{X}$ and agree on B_ν, they are identical as polynomials by the uniqueness of multilinear extensions. Hence $W(\boldsymbol{X}, Y) - W_r(\boldsymbol{X})$ vanishes identically at $Y = r$ and is divisible by $(Y - r)$, so a valid quotient polynomial $Q(\boldsymbol{X}, Y)$ exists. The quotient check succeeds.
2. The polynomial $F(\boldsymbol{X}) := W_r(\boldsymbol{X}) \cdot T(\boldsymbol{X})$ corresponds to the batched linear constraint. Since the constraints hold, $\sum_{\boldsymbol{x} \in B_\nu} F(\boldsymbol{x}) = 0$.
3. By the perfect completeness of the Sumcheck protocol (Theorem 2), the verifier accepts the sumcheck proof for $(0; [[F]]; F) \in \mathcal{R}_{\mathsf{sum}}$.

Thus, the honest prover always convinces the verifier.
Knowledge soundness. By Lemma 4, it suffices to bound the soundness error against a cheating prover. We consider three failure events where the verifier accepts despite the relation not holding:

- $\mathsf{E}_{\mathsf{Quot}}$: The quotient check passes, but $W_r(\boldsymbol{X}) \not\equiv W(\boldsymbol{X}, r)$.
- $\mathsf{E}_{\mathsf{Sum}}$: The sumcheck protocol accepts, but $\sum_{\boldsymbol{x}} W_r(\boldsymbol{x}) T(\boldsymbol{x}) \neq 0$.
- $\mathsf{E}_{\mathsf{Lin}}$: $W_r \equiv W(\cdot, r)$ and $\sum W_r T = 0$, but the original linear constraints are violated.

The total soundness error ϵ_l is bounded by the sum of these probabilities:

$$\epsilon_l \leq \Pr[\mathsf{E}_{\mathsf{Quot}}] + \Pr[\mathsf{E}_{\mathsf{Sum}} \mid \neg \mathsf{E}_{\mathsf{Quot}}] + \Pr[\mathsf{E}_{\mathsf{Lin}} \mid \neg \mathsf{E}_{\mathsf{Quot}}].$$

1. Bound $\Pr[\mathsf{E}_{\mathsf{Quot}}]$: If $W_r(\boldsymbol{X}) \neq W(\boldsymbol{X}, r)$, then $D(\boldsymbol{X}, Y) := (Y - r)Q(\boldsymbol{X}, Y) - (W(\boldsymbol{X}, Y) - W_r(\boldsymbol{X}))$ is a non-zero polynomial of degree at most $(\nu + w)$. By the Schwartz-Zippel lemma, evaluating D at random $(\boldsymbol{\alpha}, \gamma)$ yields 0 with probability at most $\frac{\nu + w}{|\mathbb{F}|}$.
2. Bound $\Pr[\mathsf{E}_{\mathsf{Sum}} \mid \neg \mathsf{E}_{\mathsf{Quot}}]$: If the quotient check holds, the verifier runs sumcheck on $F = W_r \cdot T$. The degree of F is at most 2 in each variable (since W_r and T are multilinear). By the soundness of the sumcheck protocol (Theorem 2), the error is at most $\frac{2\nu}{|\mathbb{F}|}$.

3. Bound $\Pr[\mathsf{E}_{\mathsf{Lin}} \mid \neg \mathsf{E}_{\mathsf{Quot}}]$: Suppose $\sum_{\boldsymbol{x}} W(\boldsymbol{x}, r)T(\boldsymbol{x}) = 0$. Let $C_t := \sum \beta_i^{(t)} T(\boldsymbol{b}_i^{(t)})$. The batched check is equivalent to checking if the univariate polynomial $P(Y) = \sum_{t=0}^{w-1} C_t Y^t$ evaluates to 0 at $Y = r$. If the constraints are violated, at least one $C_t \neq 0$, so $P(Y)$ is non-zero with degree at most $w-1$. The probability that $P(r) = 0$ is at most $\frac{w}{|\mathbb{F}|}$.

Summing these up:

$$\epsilon_l \leq \frac{\nu + w}{|\mathbb{F}|} + \frac{2\nu}{|\mathbb{F}|} + \frac{w}{|\mathbb{F}|} = \frac{3\nu + 2w}{|\mathbb{F}|}.$$

A.3 Proof of Theorem 5

Proof. **Completeness.** For any $(\mathcal{Q}; \{S_i\}_{i=0}^{\ell_{\mathsf{q}}-1}; \mathsf{PI}; T) \in \mathcal{R}_{\mathsf{pcake}}$, by Definition 6, it holds that

- $([[\mathsf{G}]]; \mathsf{G}) \in \mathcal{R}_{\mathsf{zero}}$, by perfect completeness of Π_{zero} (Theorem 3), $\mathcal{V}$ accepts this sub-protocol.
- $(\mathcal{Q}; [[T]]; T) \in \mathcal{R}_{\mathsf{linear}}$, by perfect completeness of Π_{linear} (Theorem 4), $\mathcal{V}$ accepts this sub-protocol.

Since all sub-protocols are perfectly complete, the full protocol is perfectly complete.

Knowledge soundness. If the tuple is not in $\mathcal{R}_{\mathsf{pcake}}$, then either the Gate Check fails or the Linear Check fails.

- Gate Check Failure: If the gate constraints are not satisfied, G is not identically zero. By Theorem 3, the ZeroCheck PIOP accepts with probability at most $O(\frac{d\nu}{|\mathbb{F}|})$.
- Linear Check Failure: If the linear constraints are not satisfied, then by Theorem 4, instantiated with the $(\nu + \nu_{\mathsf{w}})$-variate witness T, Π_{linear} accepts with probability at most $\frac{2w+3(\nu+\nu_{\mathsf{w}})}{|\mathbb{F}|}$.

Combining these, the total soundness error ϵ_p is bounded by the maximum of the two errors:

$$\epsilon_p \leq O\left(\frac{d\nu + w}{|\mathbb{F}|}\right).$$

Complexity Analysis.

- Query complexity: $2\nu + \nu_{\mathsf{w}} + 4$. (Zerocheck over B_ν: $\nu + 1$, sumcheck for the linear check over $B_{\nu+\nu_{\mathsf{w}}}$: $\nu + \nu_{\mathsf{w}} + 1$, W_r and Q: 2).
- Round complexity: $2\nu + \nu_{\mathsf{w}} + 3$. (Witness: 1, zerocheck sumcheck over B_ν: ν, linear sumcheck over $B_{\nu+\nu_{\mathsf{w}}}$: $\nu + \nu_{\mathsf{w}}$, W_r and Q: 2).
- Proof Oracle Size: $O(w \cdot N)$ (dominated by $[[Q]]$).
- Prover Time: $O(w \cdot N)$. Computing Q takes $O(w \cdot N)$.
- Verifier Time: $O(\nu)$. Dominated by $O(\nu)$ rounds of sumcheck verification and a constant number of PCS pairing checks.

A.4 Augmented-SRS extraction and separation

The prover key contains multilinear basis commitments, circuit-specific commitments involving $(\boldsymbol{\tau}, \tau_y)$, and univariate powers involving the independent trapdoor s. An algebraic prover may combine elements from all three parts. For each public $\mathbb{G}_1$ element, we associate its defining exponent polynomial in the formal variables $(\boldsymbol{X}, Y, S)$. Thus, the multilinear basis elements correspond to $\mathsf{eq}(\boldsymbol{b}, \boldsymbol{X})$, the circuit-specific elements correspond to their defining polynomials in $(\boldsymbol{X}, Y)$, and the univariate powers correspond to $1, S, \ldots, S^{d-1}$. The AGM representation of every prover-generated group element therefore determines a supported formal polynomial.

For integers m, n, the (m, n)-DLOG problem asks to recover a uniformly sampled $\chi \in \mathbb{F}^\times$ from $\{[\chi^i]_1\}_{i=0}^m$ and $\{[\chi^i]_2\}_{i=0}^n$. We use the standard assumption that this problem is hard for polynomially bounded m and n.

Let

$$D_{\mathsf{aug}} := \nu + w + d + 2$$

be a uniform upper bound on the total degree of every formal polynomial arising from the public exponents and from multiplying a prover-generated $\mathbb{G}_1$ representation by one of the linear $\mathbb{G}_2$ factors used in verification.

Lemma 5 (Formal lifting for the augmented SRS). *Assume the $(D_{\mathsf{aug}}, D_{\mathsf{aug}})$-DLOG assumption in the bilinear groups. For any algebraic prover, except with probability*

$$\epsilon_{\mathsf{lift}} := \mathsf{Adv}^{(D_{\mathsf{aug}}, D_{\mathsf{aug}})\text{-}\mathsf{DLOG}}(\lambda) + \frac{D_{\mathsf{aug}}}{|\mathbb{F}|},$$

every accepted quotient, multilinear-opening, and univariate-opening equation implies the corresponding formal polynomial identity obtained by replacing $(\boldsymbol{\tau}, \tau_y, s)$ with $(\boldsymbol{X}, Y, S)$.

The proof follows the standard AGM reduction for KZG with an extended SRS: all independent trapdoors are embedded as random affine functions of one DLOG challenge, and a false formal identity yields a nonzero univariate polynomial having the hidden discrete logarithm as a root.

Proof. Let $\mathcal{A}$ be an algebraic prover and let the AGM extractor record the coefficient representation of every group element output by $\mathcal{A}$. Replacing each public $\mathbb{G}_1$ element by its defining exponent polynomial gives, for every prover-generated element C, a supported polynomial $c_C(\boldsymbol{X}, Y, S)$. The quotient verifier uses the linear factor $Y - r$, the multilinear verifier uses the factors $X_i - a_i$, and the univariate verifier uses the factor $S - a_i$. Consequently, every accepted verifier equation determines a formal residual polynomial $P(\boldsymbol{X}, Y, S)$ of total degree at most D_{aug} such that

$$P(\boldsymbol{\tau}, \tau_y, s) = 0.$$

It remains to show that an algebraic prover cannot make this equality hold for a nonzero P except with the stated probability.

Suppose that this happens with probability ε. We construct an algorithm $\mathcal{B}$ for the $(D_{\mathsf{aug}}, D_{\mathsf{aug}})$-DLOG problem. Given powers of a hidden $\chi \in \mathbb{F}^{\times}$ in both source groups, $\mathcal{B}$ samples independent affine masks

$$\tau_i = \rho_i \chi + \theta_i, \qquad \tau_y = \rho_y \chi + \theta_y, \qquad s = \rho_s \chi + \theta_s,$$

with all ρ and θ uniform in $\mathbb{F}$. Since every exponent polynomial in the augmented SRS has total degree at most D_{aug}, the corresponding group elements can be generated from the supplied powers of χ. The resulting SRS is distributed exactly as an honestly generated one because the affine images above are mutually independent and uniform.

From a false accepted equation and its AGM representations, $\mathcal{B}$ obtains a nonzero residual P and forms

$$L(Z) := P(\rho_1 Z + \theta_1, \ldots, \rho_\nu Z + \theta_\nu, \rho_y Z + \theta_y, \rho_s Z + \theta_s).$$

The polynomial L has degree at most D_{aug} and satisfies $L(\chi) = 0$. We next justify the independence needed below. Fix any $\chi \in \mathbb{F}^{\times}$ and write $\boldsymbol{u} := (\tau_1, \ldots, \tau_\nu, \tau_y, s)$, with the masks collected into $\boldsymbol{\rho}, \boldsymbol{\theta} \in \mathbb{F}^{\nu+2}$. The map

$$(\boldsymbol{\rho}, \boldsymbol{\theta}) \longmapsto (\boldsymbol{u}, \boldsymbol{\theta}) = (\chi\boldsymbol{\rho} + \boldsymbol{\theta}, \boldsymbol{\theta})$$

is a bijection on $\mathbb{F}^{2(\nu+2)}$, with inverse $\boldsymbol{\rho} = (\boldsymbol{u} - \boldsymbol{\theta})/\chi$. Hence $\boldsymbol{u}$ and $\boldsymbol{\theta}$ are independent uniform vectors, even conditioned on χ. The simulated SRS is a deterministic function of $\boldsymbol{u}$ and does not expose either mask vector. Therefore, after conditioning on $\boldsymbol{u}$, all prover and verifier coins, and the resulting nonzero residual P, the vector $\boldsymbol{\theta}$ remains uniform. If $L \equiv 0$, then in particular $L(0) = P(\boldsymbol{\theta}) = 0$. The Schwartz–Zippel lemma consequently gives

$$\Pr[L \equiv 0] \le \Pr[P(\boldsymbol{\theta}) = 0] \le \frac{D_{\mathsf{aug}}}{|\mathbb{F}|}.$$

Outside this event, $\mathcal{B}$ factors L, tests its at most D_{aug} roots against the supplied group element $[\chi]_1$, and recovers χ. Therefore

$$\varepsilon \le \mathsf{Adv}^{(D_{\mathsf{aug}}, D_{\mathsf{aug}})\text{-}\mathsf{DLOG}}(\lambda) + \frac{D_{\mathsf{aug}}}{|\mathbb{F}|},$$

which proves the lemma.

Lemma 6 (Augmented-SRS specialization and separation). *Under the assumptions of Lemma 5, let $c_{W,j}(\boldsymbol{X}, Y, S)$ and $c_{T,j}(\boldsymbol{X}, Y, S)$ be the formal representations extracted from $\hat{C}_{W,j}$ and $\hat{T}_j$. If the quotient pairing check and the final batched multilinear opening accept, then, except with probability*

$$\epsilon_{\mathsf{lift}} + \frac{\ell_{\mathsf{w}} - 1}{|\mathbb{F}|} + \frac{1}{|\mathbb{F}|} + \frac{k_{\mathsf{w}}\nu}{|\mathbb{F}| - 2},$$

the following statements hold simultaneously for every $j \in [0, k_{\mathsf{w}} - 1]$:

1. *the value used by the final sumcheck check is the intended specialization* $W_j(\boldsymbol{a}, r)$*, even if* $c_{W,j}$ *initially contains* Y*- or* S*-dependent terms;*
2. $c_{T,j}$ *is a multilinear polynomial in* $\boldsymbol{X}$ *alone and its claimed opening is* $c_{T,j}(\boldsymbol{a})$*.*

The term $(\ell_{\mathsf{w}} - 1)/|\mathbb{F}|$ *is the conservative error of the* γ*-batching step.*

Proof. Work on the lifting event of Lemma 5. The AGM extractor gives supported formal representations $c_{W,j}(\boldsymbol{X}, Y, S)$ for $\hat{C}_{W,j}$ and $q_\gamma(\boldsymbol{X}, Y, S)$ for $\hat{C}_Q$. The accepted quotient pairing equation lifts to

$$(Y - r)q_\gamma(\boldsymbol{X}, Y, S) = \sum_{j=0}^{k_{\mathsf{w}}-1} \gamma^j \big(W_j(\boldsymbol{X}, Y) - c_{W,j}(\boldsymbol{X}, Y, S)\big). \tag{8}$$

Substituting $Y = r$ gives

$$\sum_{j=0}^{k_{\mathsf{w}}-1} \gamma^j \big(W_j(\boldsymbol{X}, r) - c_{W,j}(\boldsymbol{X}, r, S)\big) = 0.$$

The residual polynomials in this equation are fixed before γ is sampled. If any residual is nonzero, the left-hand side is a nonzero polynomial in γ of degree at most $k_{\mathsf{w}} - 1$, which is at most $\ell_{\mathsf{w}} - 1$. Therefore, except with probability $(\ell_{\mathsf{w}} - 1)/|\mathbb{F}|$,

$$c_{W,j}(\boldsymbol{X}, r, S) \equiv W_j(\boldsymbol{X}, r) \tag{9}$$

for every actual wire position j.

We next use the final batched multilinear opening. Let $c_B(\boldsymbol{X}, Y, S)$ be the ζ-weighted linear combination of the formal representations of all selector, witness, and $W_{r,j}$ commitments. Polynomial lifting turns the accepted multilinear KZG equation into

$$c_B(\boldsymbol{X}, Y, S) - v_B = \sum_{i=1}^{\nu} u_i(\boldsymbol{X}, Y, S)(X_i - a_i). \tag{10}$$

After substituting $\boldsymbol{X} = \boldsymbol{a}$, the right-hand side vanishes. Each component commitment and each claimed evaluation is fixed before the independent batching coefficients ζ_i are sampled. If any component residual is a nonzero formal polynomial in (Y, S), their random linear combination is zero with probability at most $1/|\mathbb{F}|$. Outside this event, every component satisfies its own opening identity. In particular, writing $v_{W,j}$ and $v_{T,j}$ for the claimed values,

$$c_{W,j}(\boldsymbol{a}, Y, S) \equiv v_{W,j}, \qquad c_{T,j}(\boldsymbol{a}, Y, S) \equiv v_{T,j}. \tag{11}$$

Evaluating the first identity at $Y = r$ and combining it with Equation (9) gives

$$v_{W,j} = c_{W,j}(\boldsymbol{a}, r, S) = W_j(\boldsymbol{a}, r).$$

Thus hidden Y- or S-dependence in $\hat{C}_{W,j}$ cannot change the value used by the final sumcheck check.

It remains to separate the witness commitments. Expand the supported representation of $\hat{T}_j$ as

$$c_{T,j}(\boldsymbol{X}, Y, S) = t_{j,0,0}(\boldsymbol{X}) + \sum_{(u,v)\neq(0,0)} t_{j,u,v}(\boldsymbol{X}) Y^u S^v.$$

Every coefficient $t_{j,u,v}$ is multilinear in $\boldsymbol{X}$ because every public exponent polynomial is multilinear in those variables. The second identity in Equation (11) is a scalar identity, so every nonconstant coefficient must satisfy $t_{j,u,v}(\boldsymbol{a}) = 0$. The commitment $\hat{T}_j$ is fixed before the sumcheck challenges $\boldsymbol{a}$ are sampled. If any such coefficient is nonzero, the Schwartz–Zippel lemma over $(\mathbb{F} \setminus \{0,1\})^\nu$ bounds this event by $\nu/(|\mathbb{F}| - 2)$. A union bound over the k_{w} actual witness commitments gives $k_{\mathsf{w}}\nu/(|\mathbb{F}| - 2)$. Outside this event, $c_{T,j}$ is the multilinear polynomial $t_{j,0,0}(\boldsymbol{X})$ alone and $v_{T,j} = t_{j,0,0}(\boldsymbol{a})$.

Adding the formal-lifting failure, the γ-batching failure, the ζ-batching failure, and the witness-separation failure proves the lemma.

A.5 Proof of Theorem 6

Proof. **Completeness.** Let T satisfy $\mathcal{R}_{\mathsf{pcake}}$. For every actual wire position j, the polynomial identity

$$W_j(\boldsymbol{X}, Y) - W_j(\boldsymbol{X}, r) = (Y - r) Q_j(\boldsymbol{X}, Y)$$

holds, where $Q_j = \sum_{i=0}^{w-2} r^i p_i^{(j)}$. By additivity of the commitments, the batched pairing equation therefore holds for every γ. The two summands in the combined sumcheck have zero hypercube sum. The first vanishes because all linear constraints hold, and the second vanishes because all gate constraints hold on B_ν. The committed sumcheck is perfectly complete, and the univariate and multilinear KZG openings verify by correctness of the two commitment schemes. Hence the verifier always accepts.

Knowledge soundness. Condition on the witness-extended-emulation event and apply Lemma 6. The extractor obtains multilinear witness polynomials $T_0, \ldots, T_{k_{\mathsf{w}}-1}$ in $\boldsymbol{X}$ alone. The same lemma shows that every value attributed to $\hat{C}_{W,j}$ in the final check is exactly $W_j(\boldsymbol{a}, r)$, rather than an evaluation of an adversarial representation at the hidden setup value τ_y.

By Lemma 5, an accepted univariate opening gives a formal identity of the form $c_i(\boldsymbol{X}, Y, S) - v_i = (S - a_i)\pi_i(\boldsymbol{X}, Y, S)$. Because the public $\mathbb{G}_1$ exponents are a linear union of polynomials in $(\boldsymbol{X}, Y)$ and powers $1, S, \ldots, S^{d-1}$, both c_i and π_i contain no mixed monomials involving S together with $\boldsymbol{X}$ or Y. Comparing the coefficients of the mixed terms on the right forces the $(\boldsymbol{X}, Y)$-dependent part of π_i to vanish, and then forces the $(\boldsymbol{X}, Y)$-dependent part of c_i to vanish as well. Thus every accepted round commitment is bound to a univariate polynomial $q_i(S)$ of degree at most $d - 1$. The verifier checks $g_1(0) + g_1(1) = 0$. For $i < \nu$,

it defines $h_i = g_{i+1}(0) + g_{i+1}(1)$ from the next-round endpoints, while h_ν is the final evaluation of the combined integrand. The univariate batch opening pins the verifier-derived value

$$q_i(a_i) = \frac{h_i - L_i(a_i)}{a_i(a_i - 1)}$$

and therefore enforces $g_i(a_i) = h_i$ for every $i \in [\nu]$. To justify these divisions without changing the protocol's challenge distribution, condition on the event that $a_i \notin \{0, 1\}$ for every i. Its complement has probability at most $2\nu/|\mathbb{F}|$. Conditioned on this event, the accepted transcript satisfies the complete sumcheck chain, and committed-sumcheck soundness fails with probability at most $\nu(d+1)/(|\mathbb{F}| - 2)$.

For the gate part, define

$$Z_{\mathsf{gate}}(\boldsymbol{z}) := \sum_{\boldsymbol{b} \in B_\nu} \mathsf{G}(\boldsymbol{b})\mathsf{eq}(\boldsymbol{b}, \boldsymbol{z}).$$

If any gate constraint is violated, Z_{gate} is a nonzero multilinear polynomial, so it vanishes at random $\boldsymbol{z}$ with probability at most $\nu/|\mathbb{F}|$. For the linear part, let

$$C_t := \sum_i \beta_i^{(t)} T(\boldsymbol{b}_i^{(t)}), \qquad P(r) := \sum_{t=0}^{w-1} C_t r^t.$$

The witness commitments precede r. If any linear constraint is violated, P is a fixed nonzero polynomial of degree at most $w - 1$, so $P(r) = 0$ with probability at most $(w-1)/|\mathbb{F}|$. Finally, condition on the independently sampled $\boldsymbol{z}$. The challenge α is uniform and independent, so an invalid gate sum and an invalid linear sum cancel in their random linear combination with probability at most $1/|\mathbb{F}|$.

Lemma 6 contributes

$$\epsilon_{\mathsf{lift}} + \frac{\ell_{\mathsf{w}} - 1}{|\mathbb{F}|} + \frac{1}{|\mathbb{F}|} + \frac{k_{\mathsf{w}}\nu}{|\mathbb{F}| - 2}.$$

Adding the witness-extended-emulation error, the excluded endpoint event, committed sumcheck, the gate test, the r-batching step, and the α-batching step gives

$$\epsilon_{\mathsf{lift}} + \epsilon_{\mathsf{wee}} + \frac{w + \ell_{\mathsf{w}} + 3\nu}{|\mathbb{F}|} + \frac{\nu(d + 1 + k_{\mathsf{w}})}{|\mathbb{F}| - 2}.$$

The extractor forms the canonical zero-padded multilinear encoding T from the actual slices $T_0, \ldots, T_{k_{\mathsf{w}}-1}$. The preceding argument shows that T satisfies $\mathcal{R}_{\mathsf{pcake}}$.

A.6 Proof of Corollary 1

Proof. By Theorem 6, the interactive protocol is a perfectly complete argument of knowledge in the Algebraic Group Model. The protocol is public coin, and the transcript order in Section 5.1 absorbs each prover message before deriving

the corresponding verifier challenge. Moreover, the hash-to-field procedure reproduces the challenge distribution of the interactive protocol. Applying the multi-round Fiat–Shamir transformation of [1,49] therefore yields a non-interactive argument of knowledge in the Random Oracle Model. Succinctness follows from the communication complexity of the interactive protocol.

B Performance comparison with Groth16 and Garuda

To situate Pancake among SNARKs with circuit-specific setups, we benchmark it against Groth16 [23] (implementation: Arkworks[3]) and Garuda [14] (implementation: Garuda-pari[4]), using the Rescue hash function [44]. Our benchmark circuit, taken from the Jellyfish library[5], has an addition-gate ratio of 35%.

The systems differ fundamentally in arithmetization and proving complexity. Groth16 (R1CS-based) has a quasi-linear-time prover (due to FFTs) and requires more constraints for the same operation. In contrast, Pancake (Plonkish) and Garuda (GR1CS-based) both achieve linear-time proving and are more constraint-efficient by leveraging support for high-degree custom gates. Our single-threaded experimental results are as follows:

Table 5. Proving times (seconds) for Groth16, PANCAKE, and Garuda on varying numbers of Rescue hash instances (single thread).

# Rescue Instances	Groth16	Pancake	Garuda
2^5	0.54	0.21	0.21
2^6	1.00	0.36	0.29
2^7	1.90	0.63	0.52
2^8	3.45	1.16	0.93
2^9	6.51	2.14	1.66
2^{10}	12.00	4.01	3.11
2^{11}	22.74	7.55	5.93
2^{12}	42.75	14.33	11.11

As shown in Table 5, both Pancake and Garuda significantly outperform Groth16. This result demonstrates that Pancake attains prover efficiency comparable to the recent circuit-specific SNARK Garuda, while remaining within the Plonkish arithmetization framework and preserving its expressiveness for custom gates.

[3] https://github.com/arkworks-rs/groth16
[4] https://github.com/alireza-shirzad/garuda-pari
[5] https://github.com/EspressoSystems/jellyfish