Architecture-private Zero-knowledge Proof of Neural Networks

Yanpei Guo
National University of Singapore
guo.yanpei@u.nus.edu

Zhanpeng Guo
Xidian University
zhanp.guo@gmail.com

Wenjie Qu
National University of Singapore
wenjiequ@u.nus.edu

Jiaheng Zhang
National University of Singapore
jhzhang@nus.edu.sg

***Abstract*—A zero-knowledge proof of machine learning (zkML) enables a party to prove that it has correctly executed a committed model using some public input, without revealing any information about the model itself. An ideal zkML scheme should conceal both the model architecture and the model parameters. However, existing zkML approaches for neural networks primarily focus on hiding model parameters. For convolutional neural network (CNN) models, these schemes reveal the entire architecture, including number and sequence of layers, kernel sizes, strides, and residual connections.**

In this work, we initiate the study of architecture-private zkML for neural networks, with a focus on CNN models. Our core contributions includes 1) parametrized rank-one constraint system (pR1CS), a generalization of R1CS, allowing the prover to commit to the model architecture in a more friendly manner; 2) a proof of functional relation scheme to demonstrate the committed architecture is valid.

Our scheme matches the prover complexity of BFG+23 (CCS'23), the current state-of-the-art in zkML for CNNs. Concretely, on VGG16 model, when batch proving 64 instances, our scheme achieves only 30% slower prover time than BFG+23 (CCS'23) and 2.3× faster than zkCNN (CCS'21). This demonstrates that our approach can hide the architecture in zero-knowledge proofs for neural networks with minor overhead. In particular, proving a matrix multiplication using our pR1CS can be at least 3× faster than using conventional R1CS, highlighting the effectiveness of our optimizations.

1. Introduction

Zero-knowledge succinct argument of knowledge (zkSNARK) [1], [2], [3], [4] allows a prover to convince a verifier that a statement is true, without revealing any information beyond the statement's validity. Formally speaking, given a *public* arithmetic circuit $\mathcal{C}$ and a public instance $\mathbb{x}$, the prover demonstrates the statement that it possesses a witness $\mathbb{w}$ such that $\mathcal{C}(\mathbb{x}, \mathbb{w})$ is fully satisfied, without revealing $\mathbb{w}$. Succinct means the *verifier time* and *proof size* can be exponentially smaller than circuit size. Typically, zkSNARKs for general arithmetic circuits [1], [4], [5] use a precessing algorithm to digest the large circuit $\mathcal{C}$ into a short verifier parameter vp, and thus verification algorithm can avoid reading the entire circuit.

Zero-knowledge machine learning (zkML) [6], [7], [8] is an important application of zkSNARK that enables privacy-preserving regulation for *Machine Learning-as-a-Service* (MLaaS) [9], [10], where users are usually given an API of AI models and receive a nontransparent result from the model server. Employing zkML to prove the inference result can guarantee the following properties:

- **Integrity**: prevent the service provider from using distilled version of the model to reduce inference costs.
- **Fairness**: exclude sensitive user attributes such as gender, race, or past behavior from the decision-making process.
- **Privacy**: preserve model owner's intellectual property as the model is completely a blackbox.

Formally speaking, let *deterministic* function f_p denote the inference model. In an ideal zkML, model owner publish a private commitment to function f_p, and invokes a zkSNARK to prove $y = f_p(x, r)$ for any input x, randomness r and output y.

With the widespread adoption of machine learning, particularly neural networks, recent years have seen extensive research on zkML. This research spans a variety of models, including traditional decision trees [6], [11], convolutional neural networks [7], [12], [13], and large language models [8], [14], [15]. In typical zkSNARK schemes for neural networks, *the model architecture is encoded into a circuit* $\mathcal{C}$, while the model parameters p are represented as polynomials and committed using a polynomial commitment scheme (PCS). For each machine learning prediction $f_p(x) = y$, the server proves that $\mathcal{C}(p, x) = y$.

Private neural network architecture. A limitation of existing zkML [7], [8], [13] approaches is that they only considers hiding model parameters, but assumes that the model architecture is completely public. In practice, model architectures also constitute valuable intellectual properties for model owners [16], [17], as designing effective architectures demands a team of highly skilled experts, extensive trial and error, significant financial costs, and substantial

computational resources. This motivates us to develop an zkML scheme hiding both model parameter and architecture.

Hiding model architecture is highly challenging. As conventional zkML designs typically encode architecture into an arithmetic circuit, and zkSNARK requires the circuit to be publicly known, it's difficult to keep the model architecture confidential. What's more, extensive zkML constructions leverage structural ML architecture to improve efficiency [7], [15], making it more challenging to achieve comparable efficiency without publishing the architecture.

In this work, we address these challenges and initiate the study of *architecture-private* zero-knowledge proofs for neural networks. We propose an efficient construction specifically tailored for CNN models. With our design, users can verify that the model prediction is computed by a precommitted function f_p, without revealing any information about its internal architecture[1]. Given that CNN architectures can vary significantly in terms of the number of layers, kernel sizes, strides, and microstructures such as the residual connections in ResNet [18], we believe our work is both necessary and well-motivated.

We summarize our goal of architecture-private ZKP for neural network with the following research question:

> *Can we design a zkSNARK for CNN with (i) a private architecture and (ii) comparable prover time as its public-architecture counterpart?*

1.1. Our Methods

To conceal the architecture of the CNN model (i.e., circuit $\mathcal{C}$), a natural idea is to employ a general-purpose proof system, where $\mathcal{C}$ is digested into a short commitment vp. However, this naive approach faces two major challenges. The first challenge is the inefficiency of preprocessed zkSNARKs, which has been abandoned by state-of-the-art zkML schemes. For instance, proving VGG16 in Plonk takes 637 seconds [19], whereas zkCNN [7] only requires 88 seconds. The second and more critical challenge is *output non-determinism* of committed circuit $\mathcal{C}$. Specifically, an arithmetic circuit $\mathcal{C}$ represents a NP relation instead of deterministic function. It means each public input can be satisfied by multiple witnesses. Thus, when $\mathcal{C}$ is kept fully private, there may exist multiple valid outputs for the same input, undermining both integrity and fairness guarantees.

In order to deal with the inefficiency issue, we propose parametrized rank-one constraint system (pR1CS) and parametrized Spartan by generalizing conventional R1CS and its proof system Spartan. The reason why preprocessed zkSNARK is less efficient primarily lies in two following factors. The first factor is they have substantial redundant computation to deal with the general circuit. The second factor is they have to commit substantial intermediate values when executing Freivalds', the widely-employed matrix multiplication verification algorithm.

We notice that in R1CS, the first inefficiency can be mitigated by batch-proving multiple instances, as the shared computational overhead introduced by the circuit—e.g. opening a sparse polynomial $\tilde{A}(\vec{r}_x, \vec{r}_y)$ in Spartan—can be amortized. The second issue is more challenging, as Freivalds' algorithm requires $O(n^2)$ multiplication gates to compute the product between a matrix and a random vector, this will incur substantial commitment cost. Our key observation to address this challenge is that we can introduce randomness directly into the circuit descriptions A, B, and C, enabling the multiplication between a matrix row and a random vector to be expressed with a single gate.

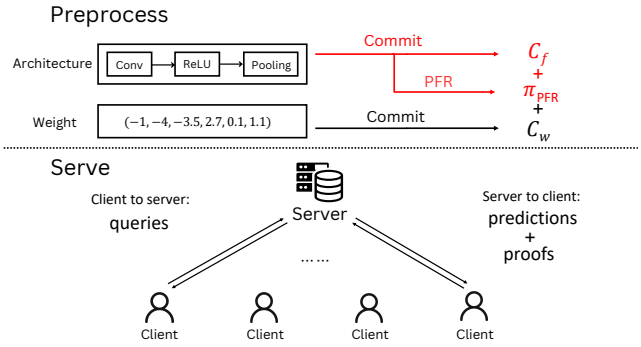

Figure 1: Framework for architecture-private zkML

To address the problem of output non-determinism, we introduce a proof of functional relation (PFR) scheme specifically designed for pR1CS and CNN models. This paradigm builds on the framework of BNO21 [20], where the prover attaches a proof attesting that the committed circuit corresponds to an output-binding function. However, BNO21 assumes circuits are computed gate-by-gate, which precludes efficient verification methods such as Freivalds' algorithm. BNO21 also doesn't consider lookup gates. To overcome these limitations, we design an simple yet expressive *pR1CS-compatible virtual machine* with only four instructions. In this setting, the prover only needs to demonstrate that the circuit is a correct compilation of a program, which suffices to certify its determinism.

Figure 1 presents the framework of our amortized architecture-private zkML. We summarize our contributions as follows:

1) **Architecture-private zkML**: We provide the first architecture-private zkSNARK for CNN. It leaks nothing information about the model architecture, except an approximate scale. The prover complexity is identical to BFG+23 [13], the state-of-the-art scheme for verifiable CNN. Concretely, our scheme achieves comparable prover efficiency with BFG+23 [13] when batch proving multiple instances.
2) **Parametrized rank-one constraint system**: We generalize R1CS into a *parametrized* version, referred to as pR1CS, which may be of independent interest. Compared to conventional R1CS, verifying the multiplication of two $n \times n$ matrices in pR1CS requires only $O(n)$ gates, whereas traditional R1CS requires at least $3n^2$ gates. We also extend Spartan to support proving circuits in the pR1CS format.

1. The only leaked information about f_p is its approximate scale.

3) **Implementation**: We implemented our architecture-private zkML scheme and instantiated it with VGG11 and VGG16. We compare the performance of our architecture-private scheme with existing architecture-public schemes. When batching the proof over 64 images, our scheme is only 30-40% slower than BFG+23 and 2.3× faster than zkCNN [7]. We also benchmark matrix multiplication using pR1CS and conventional R1CS, and find that proving matrix multiplication in pR1CS is about 3-4× faster than in conventional R1CS, demonstrating the efficiency of our proposed constraint system.

1.2. Related Works

1.2.1. Zero-knowledge machine learning. Substantial research has been conducted on developing zkSNARKs for machine learning. These efforts target various models, including decision trees [6], [11], CNNs [7], [12], [13] and LLMs [8], [14]. In terms of proof systems, prevalent zkML schemes can be broadly categorized into three main approaches: VOLE-based [14], [21], GKR-based [7], [13], and customized SNARKs [8]. Each scheme presents different trade-offs between prover time and communication cost, but all neural network schemes[2] require the model architecture to be publicly available.

Verifiable CNN. Due to the importance and prevalence of CNNs, there has been substantial research interest in verifiable CNNs. Early works [12], [19], [22] incur substantial overhead, which require several hours or even days to generate proofs for CNN inference. These works are constructed directly based on zkSNARKs for general circuits written in R1CS [12], [22] or Plonkish [19]. A significant milestone in verifiable CNN is zkCNN [7] (CCS'21), which was the first to develop a specialized protocol for proving convolutional layers, leveraging efficient proofs based on zkFFT [7]. More recently, BFG+23 [13] (CCS'23) proposes a more efficient scheme for convolutional layers, by unrolling convolution operations into matrix multiplications. It remains the state-of-the-art scheme in terms of prover complexity of verifiable CNN, and has a succinct communication cost.

1.2.2. Functional commitment. The research goal that committing a private CNN inference model aligns with *functional commitment*. As pointed out in [20], previous functional commitments have dual meanings. The first is *input-hiding* functional commitments [23], [24], [25], [26]: the committer commits to a private input $\vec{x}$ and later proves that $f(\vec{x}) = y$ for some public function f and a value y.

In our paper, we primarily focus on the second case, known as *function-hiding* functional commitment [20]. In this setting, the committer commits to a private *output-binding* function f and later proves that $f(\vec{x}) = y$ for some public input $\vec{x}$ and output y. Function-hiding functional commitments were introduced by BNO21 [20], with constructions based on preprocessing zkSNARKs such as Marlin [5] and Plonk [1]. Several subsequent works [27], [28] have improved upon BNO21 from a theoretical perspective, aiming to eliminate reliance on the *random oracle* model. Most of these approaches are based on lattice-based assumptions and leverage techniques from fully homomorphic encryption, which tend to be less efficient in practice.

1.2.3. Customizable Constraint System. A limitation of R1CS is that it only supports "rank-1" constraints, i.e. the degree of each gate is at most 2. *Customizable Constraint System* (CCS) [29] is a generalization of R1CS to support *high-degree* customized gates. Although CCS is a significant advancement over R1CS in general workload, high-degree gates are not particularly useful for zkML.

Our proposed constraint system, pR1CS, aims to address another limitation of R1CS: its inability to express the inner product of two vectors within a single constraint. Specifically, to compute the inner product of two variable vectors $\vec{a}, \vec{b}$, R1CS must introduce an intermediate vector $\vec{c} = \vec{a} \circ \vec{b} \in \mathbb{F}^n$, and then compute that $s = \sum_{i \in [n]} \vec{c}[i]$.

2. Preliminary

Notation. We use $[n]$ to denote the vector $(0, 1, \cdots, n-1)$, and $[a, b)$ to denote the vector $(a, a+1, \cdots, b-1)$. We use $\vec{v}[i]$ to denote the i-th element of the vector $\vec{v}$. By default, for a vector $\vec{v}$ of length n, we let $\vec{v}[-i]$ denote $\vec{v}[n-i]$. All vectors in this paper are assumed to be 0-indexed by default. We use $\vec{a} \circ \vec{b}$ to denote the Hadamard product of vectors, and $(\vec{a}^{(1)}, \cdots, \vec{a}^{(d)}) \circ \vec{b} = (\vec{a}^{(1)} \circ \vec{b}, \cdots, \vec{a}^{(d)} \circ \vec{b})$. We use $\langle \vec{a}, \vec{b} \rangle$ to denote inner product of vectors. For $\vec{v} \in \mathbb{F}^n$, we define the right rotation of $\vec{v}$ as $\mathsf{rsh}(\vec{v}) := (\vec{v}[n-1], \vec{v}[0], \vec{v}[1], \cdots, \vec{v}[n-2])$.

Vectors can be treated as ordered multisets. We write $a \in \vec{f}$ to indicate that there exists some i such that $a = \vec{f}[i]$, and $\vec{f} \subseteq \vec{g}$ to indicate that for every $x \in \vec{f}$, there exists $x \in \vec{g}$. We also define *sparse* vectors using a set of tuples. Specifically, we write $\vec{f} = \{(a_1, b_1), \cdots, (a_m, b_m)\} \subset \mathbb{N} \times \mathbb{F}$ to mean that $\vec{f}[a_i] = b_i$ for each i, and $\vec{f}[x] = 0$ for any $x \notin \{a_1, \cdots, a_m\}$. For a matrix A, we use $A[i]$ to denote its i-th row.

2.1. Functional Commitment

Definition 1 (Functional Commitment)**.** *A functional commitment scheme (FCS)* FC *consists of a tuple of algorithms. It commits a function $f \in \mathcal{F}$, where $\mathcal{F}$ is the set of all function candidates.*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{gp}$. *It takes the security parameter λ and outputs the global parameter* gp.
- $\mathsf{Commit}(\mathsf{gp}, f) \to (\mathcal{C}, \mathcal{D})$. *It takes a function $\tilde{f} \in \mathcal{F}$ and outputs a commitment* com *along with some auxiliary message $\mathcal{D}$ (e.g., some randomness).*
- $\mathsf{PFR}(\mathsf{gp}, f, \mathcal{D}) \to \pi_f$*: It takes function f and auxiliary message $\mathcal{D}$, outputs a proof of functional relation π_f.*

2. ZKP for decision trees can conceal the model's architecture, as the tree structure is inherently more constrained and therefore easier to formalize within zero-knowledge protocols.

- $\mathsf{VFR}(\mathsf{gp}, \mathcal{C}, \pi_f) \rightarrow 0/1$*: It takes commitment of* f *and proof* π_c*, outputs a bit* $b \in \{0,1\}$.
- $\mathsf{Open}(\mathsf{gp}, \mathcal{C}, f, \mathcal{D}) \rightarrow 0/1$. *It's a deterministic algorithm that takes function* f*, and auxiliary message* $\mathcal{D}$ *to verify the commitment and outputs a bit* $b \in \{0,1\}$.
- $\mathsf{Eval}(\mathsf{gp}, \mathcal{D}, \vec{x}, \tilde{f}) \rightarrow (\vec{y}, \pi)$. *It takes as input auxiliary message* $\mathcal{D}$ *generated in commitment, input* $\vec{x}$ *and function* f*, outputs evaluation result* $\vec{y}$ *and evaluation proof* π.
- $\mathsf{Verify}(\mathsf{gp}, \mathcal{C}, \vec{x}, \vec{y}, \pi) \rightarrow 0/1$. *It takes as input commitment* $\mathcal{C}$*, input* $\vec{x}$*, output* $\vec{y}$ *and proof* π*, outputs a bit* $b \in \{0,1\}$.

Definition of *completeness*, *binding*, *soundness* and *hiding* of a FCS is presented in Appendix A.

This work proposes an efficient function-hiding FCS tailored for CNN. In our work, the Setup phase is used to generate the global parameters of *polynomial commitment*, while Open re-executes the Commit process using randomness $\mathcal{D}$. Other algorithms will be introduced in subsequent sections.

2.2. Multilinear extension and Sumcheck

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

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

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

The function $f : \{0,1\}^\mu \rightarrow \mathbb{F}$ can be represented as a vector $\vec{f} \in \mathbb{F}^{2^\mu}$, such that $f(\vec{b}) = \vec{f}[\mathsf{int}(\vec{b})]$, where $\mathsf{int}(\vec{b}) = \sum_{i \in [\mu]} 2^i \cdot b[i]$. Similarly, we define $\vec{eq}_{\vec{x}}$ such that $\tilde{eq}_{\vec{x}}(\vec{b}) = \vec{eq}_{\vec{x}}[\mathsf{int}(\vec{b})]$. For vector $\vec{f}$, we use $\tilde{f}(\cdot)$ to represent its MLE by default.

Polynomial Commitment Scheme. A μ-variate multilinear *polynomial commitment scheme* (PCS) is a special case of a functional commitment scheme, where $\mathcal{F} = \mathbb{F}^{(\leq 1)}[X_1, \cdots, X_\mu]$. Typically, a PCS does not require a PFR protocol, as the Eval protocol already guarantees (knowledge) soundness. There has been extensive research on constructing concretely efficient and succinct multilinear PCS [30], [31], [32], [33]. One notable property of multilinear PCS is that committing to polynomials with small scalar values over a hypercube can be significantly cheaper than committing to polynomials with arbitrary field elements.

Sumcheck Protocol. Given commitment of $\{\tilde{v}_i\}_{i=1}^d$, a sumcheck protocol can reduce the claim of

$$\sum_{i \in [2^\mu]} (\vec{v}_1 \circ \cdots \circ \vec{v}_d)[i] = y$$

into d evaluation claims over $\tilde{v}_1, \cdots, \tilde{v}_d$ as follows:

$$\tilde{v}_i(r_1, \cdots, r_\mu) = y_i$$

where r_i is the random challenge in the i-th round, shared by all vectors. The prover complexity is $O(d^2 \cdot 2^\mu)$. The verifier complexity and communication cost is $O(\mu d)$. The soundness error is $O(\frac{\mu d}{|F|})$.

2.3. Lookup Arguments and LogUp

Lookup arguments [34], [35] prove that two committed vectors $\vec{a} \in \mathbb{F}^n, \vec{b} \in \mathbb{F}^m$ satisfies $\vec{a} \subseteq \vec{b}$. Lookup arguments can be trivially extended to multi-columns, proving two vector tuples $\left(\vec{a}^{(0)}, \cdots, \vec{a}^{(c-1)}\right)$ and $\left(\vec{b}^{(0)}, \cdots, \vec{b}^{(c-1)}\right)$ satisfies $\forall i \in [n], \exists j \in [m]$, such that $\vec{a}^{(k)}[i] = \vec{b}^{(k)}[j]$ for each $k \in [c]$. This can be denoted as

$$\left(\vec{a}^{(0)}, \cdots, \vec{a}^{(c-1)}\right) \subseteq \left(\vec{b}^{(0)}, \cdots, \vec{b}^{(c-1)}\right)$$

The multi-column case can be proven through randomly combining all these columns, trading off with $\frac{c}{|\mathbb{F}|}$ soundness error, such that proving

$$\sum_{k \in [c]} \alpha^k \vec{a}^{(k)} \subseteq \sum_{k \in [c]} \alpha^k \vec{b}^{(k)}$$

where α is random challenge given by verifier.

LogUp [35], [36] is a simple and efficient construction of the lookup argument. At a high level, it proves:

$$\sum_{i \in [n]} \frac{1}{X - \vec{a}[i]} \equiv \sum_{i \in [m]} \frac{\vec{e}[i]}{X - \vec{b}[i]}$$

where $\vec{e}$ is another vector committed by prover, indicating the multiplicity of $\vec{b}[i]$ in $\vec{a}$. The above identical relation can be proved by letting the verifier sample a random element β and prove

$$\sum_{i \in [n]} \frac{1}{\beta - \vec{a}[i]} = \sum_{i \in [m]} \frac{\vec{e}[i]}{\beta - \vec{b}[i]}$$

The sumcheck of reciprocals can be verified by additional commitment to these reciprocals, or using GKR [37].

Indexed lookup. Lasso [38] introduces a primitive called indexed lookup. Given commitments of $\tilde{t}, \tilde{idx} \in \mathbb{F}^m$, $\tilde{T} \in \mathbb{F}^n$, indexed lookup requires the prover to argue that for each $i \in [m]$, $\vec{t}[i] = \vec{T}[\vec{idx}[i]]$. An indexed lookup can be easily reduced to a conventional lookup argument. More precisely, the prover only needs to prove

$$\left(\vec{t}, \vec{idx}\right) \subseteq \left(\vec{T}, [n]\right)$$

2.4. R1CS and Spartan

R1CS is a powerful way to formally capture an algebraic circuit and to translate it into a set of matrices and vectors. Specifically, the circuit is translated into three sparse matrices $A, B, C \in \mathbb{F}^{m \times n}$, where $m = \Theta(n)$. In each matrix,

only $O(n)$ elements are non-zero, so these matrices are called *sparse*. A vector $\vec{z}$ including public I/O and all *multiplication* gate outputs is committed and sent to the verifier. Specifically, for each i, $\langle A[i], \vec{z}\rangle$ and $\langle B[i], \vec{z}\rangle$ represent addition gates implicitly, while $\langle A[i], \vec{z}\rangle \cdot \langle B[i], \vec{z}\rangle = \langle C[i], \vec{z}\rangle$ represents a multiplication gate. Then, the prover only needs to prove:

$$A\vec{z} \circ B\vec{z} = C\vec{z} \tag{1}$$

Spartan [4] is a state-of-the-art preprocessing zkSNARK scheme for R1CS. During its preprocessing phase, public sparse matrices A, B, C are committed using the following idea: Each sparse matrix $M \in \mathbb{F}^{m\times n}$ can be condensed into 3 vectors $\vec{row}$, $\vec{col}$, $\vec{val}$ of length $O(n)$, representing $M[\vec{row}[i], \vec{col}[i]] = \vec{val}[i]$ and other elements in M are 0. So committing M can be transformed into committing 3 multilinear polynomials $\tilde{row}$, $\tilde{col}$, $\tilde{val}$, only requiring $O(n)$ computational time.

Spartan's proving procedure can be divided into two steps. First, prover utilizes the sumcheck protocol to reduce the claim of Equation 1 into a claim of

$$\vec{eq}_{\vec{r}_1}^T \cdot M \cdot \vec{eq}_{\vec{r}_2} = \tilde{M}(\vec{r}_1, \vec{r}_2) \stackrel{?}{=} y_M \tag{2}$$

$$\tilde{z}(\vec{r}_2) \stackrel{?}{=} y_z \tag{3}$$

where $M \in \{A, B, C\}$, $\vec{r}_1$ and $\vec{r}_2$ are two random vectors. Eq 3 can be proved by directly invoking $\mathsf{PC.Eval}$, while proving Eq 2 can be finally transformed into a series of indexed lookup arguments and sumcheck arguments.

2.5. Freivalds: Verifiable Matrix Multiplication

Freivalds' Algorithm [39] (hereinafter referred to simply as Freivalds for brevity) can be used to verify matrix multiplication $X \cdot Y = Z \in \mathbb{F}^{m\times k}$ efficiently. Specifically in terms of ZKP, when elements of X, Y and Z are all committed, verifier can sample a random $\gamma \in \mathbb{F}$ and check

$$\left(\vec{\gamma}_1^T X\right) \cdot (Y\vec{\gamma}_2) = \langle X^T\vec{\gamma}_1, Y\vec{\gamma}_2\rangle \stackrel{?}{=} \vec{\gamma}_1^T Z\vec{\gamma}_2 \tag{4}$$

where $\vec{\gamma}_1 = \left(\gamma^{0\cdot k}, \cdots, \gamma^{(m-1)\cdot k}\right)$, $\vec{\gamma}_2 = \left(\gamma^0, \cdots, \gamma^{k-1}\right)$. Assuming $m, k = \Theta(n)$, Freivalds' algorithm checks matrix multiplication in $O(n^2)$ computational time. As direct computing matrix multiplication requires $O(n^3)$, Freivalds achieves both asymptotically and concretely faster runtimes.

2.6. CNN and BFG⁺23

A Convolutional Neural Network (CNN) consists of a sequence of distinct layer types, typically including linear, activation, and pooling layers. The number and ordering of these layers can vary significantly depending on the model architecture.

Linear layer. The linear layers in a CNN include both convolutional and fully connected layers. These layers are computed via matrix multiplication between the input data and the corresponding weight matrix. In BFG⁺23, the correctness of these linear layers is verified using Freivalds' algorithm.

Activation and pooling layer. In an activation layer, a non-linear activation function is applied element-wise to the input. In a pooling layer, a kernel slides over the input matrix and computes an aggregated value from each covered region. Common pooling operations include max-pooling, which selects the maximum value in the region, and average pooling, which computes the mean.

The most efficient way to prove these layers is through lookup arguments. For example, the ReLU activation function is defined as $\mathsf{ReLU}(x) = \max(x, 0)$, so its output can be derived via a lookup table. Similarly, the max-pooling operation can be expressed as $\max(a, b) = a + \mathsf{ReLU}(b - a)$, which can also be evaluated using lookup gates.

2.6.1. Quantization. Most zkML frameworks adopt quantization to convert real-valued numbers into integers, simplifying the process of generating proofs. We follow the quantization method from [7], where a real number x is mapped to an integer q using the formula $q = \lfloor x \cdot 2^Q \rfloor$, with Q being a positive integer that defines the quantization scale.

Under this scheme, all model parameters and intermediate values are represented as B-bit integers. The value of B is chosen as the smallest integer such that for all real values x appearing during inference, the condition $-2^{B-1} \le x \cdot 2^Q < 2^{B-1}$ holds.

For addition, we simply add the quantized integers: $x + y = \frac{q_x + q_y}{2^Q}$. For multiplication, where $x = \frac{q_x}{2^Q}$ and $y = \frac{q_y}{2^Q}$, a *downscaling* step is required:

$$q_z = \left\lfloor \frac{q_x}{2^Q} \cdot \frac{q_y}{2^Q} \cdot 2^Q \right\rfloor = \left\lfloor \frac{q_x \cdot q_y}{2^Q} \right\rfloor.$$

This method allows fractional addition and multiplication to be approximated using only integer arithmetic.

3. Technical Overview

In this section, we present an overview of our architecture-private zkML scheme. Our high-level idea is committing architecture, which has been encoded into circuit $\mathcal{C}$, via $\mathsf{vp} \leftarrow \mathsf{Prep}(\mathcal{C})$, where Prep is the preprocessing algorithm in general proof system to digest large circuit into short verification key. Our techniques deal with the inefficiency and output non-determinism issue.

In Section 3.1, we introduce the *parametrized R1CS* to improve efficiency of general proof system on the task of zkML. In Section 3.2, we describe how to prove that a committed pR1CS circuit corresponds to an output-binding function. Combining the schemes introduced in these two sections can derive the end-to-end construction of architecture-private zkSNARK for CNN models.

3.1. Parametrized R1CS

Before presenting our optimization, we first analyze the prover cost of constructing zkML using the classic

R1CS approach. As discussed in Section 2.4, proving an R1CS statement with Spartan involves multiple steps, among which the evaluation of the sparse polynomials $\tilde{A}$, $\tilde{B}$, and $\tilde{C}$ (Eq. 2) accounts for nearly 90% of the total prover time (see Figure 7 in [4]). Fortunately, this cost can be *amortized* when proving multiple instances, as all instances share the same sparse polynomials—meaning that a single proof suffices for N instances. As a result, after amortization, our primary focus shifts to reducing the instance-dependent cost. In particular, we aim to minimize the cost of committing to $\vec{z}$, as this constitutes the dominant portion of the remaining overhead.

Even not considering the cost of sparse polynomial evaluation, proving a CNN circuit described in R1CS can be still costly. The main inefficiency lies in checking matrix multiplication, as R1CS requires committing many intermediate results when using Freivalds. Specifically, assume the prover wants to prove multiplications of two matrices of size $n \times n$. Computing Eq. 4 in R1CS requires more than $3n^2$ multiplication gates, as each matrix-vector multiplication requires at least n^2 multiplications. Moreover, each multiplication gate output would be a large scalar, which would significantly increase the commitment cost of $\vec{z}$. In contrast, BFG+23 can avoid any additional commitments by invoking Thaler13 [40], a sumcheck-based verifiable matrix multiplication scheme.

Parametrized R1CS. We deal with this problem by adding the random term γ used in Freivalds into sparse matrices A, B, C, instead of adding to $\vec{z}$. Specifically, the constant matrix A in R1CS transforms into a *parametrized* matrix $A(\Gamma)$, where each element may include variable Γ. An example about $A(\Gamma)$ is

$$A(\Gamma) = \begin{pmatrix} 0 & 1 & 0 \\ 2\Gamma^3 & 0 & 3\Gamma^2 \\ 4 & 0 & 0 \end{pmatrix}$$

Prover can commit sparse parametrized matrix $A(\Gamma)$ by committing four vectors:

$$\begin{aligned} r\vec{ow} = (0,1,1,2) \quad c\vec{ol} = (1,0,2,0) \\ v\vec{al} = (1,2,3,4) \quad i\vec{dx} = (0,3,2,0) \end{aligned} \tag{5}$$

which means

$$A(\Gamma)[r\vec{ow}[i], c\vec{ol}[i]] = v\vec{al}[i] \cdot \Gamma^{i\vec{dx}[i]}$$

Witness in $\vec{z}$ can be categorized into two parts: depending on Γ and independent on Γ. When committing $\vec{z}$, the prover will first commit elements independent of Γ, including all matrix multiplication inputs and outputs. After receiving random challenge $\gamma \leftarrow_{\$} \mathbb{F}$, matrices $M(\Gamma)$ become fixed into M_γ. The prover commits the remaining elements in $\vec{z}$ and finally proves

$$A_\gamma \vec{z} \circ B_\gamma \vec{z} = C_\gamma \vec{z}$$

With this technique, when proving matrix multiplication $X \cdot Y = Z$, it only requires n extra multiplication gates. For example, to check $X \cdot Y = Z$, where

$$X = \begin{pmatrix} \vec{z}[1] & \vec{z}[3] \\ \vec{z}[2] & \vec{z}[4] \end{pmatrix}, Y = \begin{pmatrix} \vec{z}[5] & \vec{z}[6] \\ \vec{z}[7] & \vec{z}[8] \end{pmatrix}, Z = \begin{pmatrix} \vec{z}[9] & \vec{z}[10] \\ \vec{z}[11] & \vec{z}[12] \end{pmatrix}$$

After receiving commitment of $\vec{z}[1..12]$, verifier can sample $\gamma \leftarrow_{\$} \mathbb{F}$, let prover commit only two values $\vec{z}[13..14]$ and check:

$$\begin{aligned} \vec{z}[13] &= \langle (1,\gamma^2), (\vec{z}[1], \vec{z}[2]) \rangle \cdot \langle (1,\gamma), (\vec{z}[5], \vec{z}[6]) \rangle \\ \vec{z}[14] &= \langle (1,\gamma^2), (\vec{z}[3], \vec{z}[4]) \rangle \cdot \langle (1,\gamma), (\vec{z}[7], \vec{z}[8]) \rangle \\ \vec{z}[13] + \vec{z}[14] &= \langle (1,\gamma,\gamma^2,\gamma^3), (\vec{z}[9], \vec{z}[10], \vec{z}[11], \vec{z}[12]) \rangle \end{aligned}$$

Compared with conventional R1CS, the primary advantage of pR1CS is that we can express the inner product between two variable vectors $\langle (1,\gamma^2), (\vec{z}[1], \vec{z}[2]) \rangle$ in one constraint by setting some row in $A(\Gamma)$ as $(0, 1, \Gamma^2, 0, \cdots)$. Using this, we reduce the cost of committing $3n^2$ multiplication gates to only n intermediate results to check a matrix multiplication.

Lookup gates. Another important building block of our proof system is *lookup gates*. In order to check that a specific element in $\vec{z}$ is equal to the lookup output of sums of other values, e.g. $\vec{z}[10] = \mathsf{ReLU}(\vec{z}[3] + \vec{z}[5])$, the R1CS lookup [41] guides us to prove

$$(D\vec{z}, E\vec{z}) \in T_{\mathsf{ReLU}} \tag{6}$$

where D, E are another two sparse matrices like A, B, C but doesn't have variables. For example, to prove $\vec{z}[10] = \mathsf{ReLU}(\vec{z}[3] + \vec{z}[5])$, we can set $D[i] = \{(3,1),(5,1)\}$ and $E[i] = \{(10,1)\}$ for some i. In order to obfuscate which activation function is used, prover has to prove multiple lookups using different public tables. This repetition of lookup could be costly.

To solve the above problem, we combine all lookups and all public tables into a larger one, using another instance-independent vector $\vec{tp}$ to represent the lookup type. Specifically, the prover proves that

$$\left(D\vec{z}, E\vec{z}, \vec{tp}\right) \subseteq T$$

where T has three columns, namely input, output and type. Similar with A, B, C, evaluation of D and E can also be amortized across multiple instances.

3.2. Proof of Functional Relation

Using the technique in Section 3.1, model server manage to encode architecture $\mathcal{C}$ into 5 sparse matrices (3 parametrized and 2 conventional) and vectors, which can be committed by PCS. However, there's a significant issue that the committed circuit may not be output-binding. That is, for a given input $\vec{x}$, there may exist two different outputs y_1 and y_2 that both satisfied by the committed circuit. For example, when prover commits a circuit presented in Figure 2, it can use w to choose the desired output.

To deal with this output non-determinism issue, BNO21 proposed to attach another proof to testify the circuit is output-binding. Specifically, BNO21 assumes witness $\vec{z}$ has been sorted based on topological order in circuit. It proves that each non-input value $\vec{z}[i]$ is computed directly from $\vec{z}[0..i-1]$ through addition or multiplication. Thus, the output $\vec{z}[n-1]$ is binding with respect to the input.

Limitation of BNO21. The major limitation of BNO21 is it restricts each non-input value in $\vec{z}$ to be computed solely

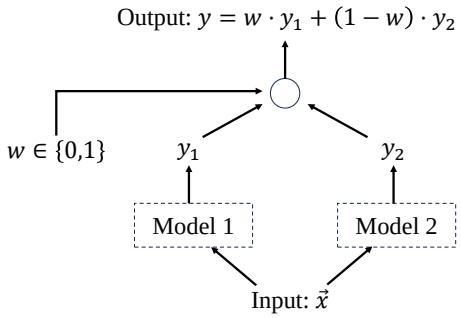

Figure 2: A dummy circuit: prover can select desired output

through addition or multiplication gates, which prevents the use of certain *efficient checkers*. For example, given a field element x as input, an efficient method [42] to compute a bit b indicating whether $x = 0$ is for the prover to additionally provide an advice input $y = x^{-1}$, prove that $x(xy-1) = 0$, and output $b = xy$. Since y is neither a public input nor computed directly from existing values through a single addition or multiplication gate, this technique is disallowed by BNO21. Similarly, commonly used ZKP techniques such as bit decomposition and Freivalds' check are also disallowed by BNO21. Consequently, functions that could otherwise benefit from these efficient checking techniques become significantly less efficient under BNO21.

The challenge to allow such *advice inputs* in proof of functional relation lies in preventing them from affecting the final output. Our key insight is that although verifying whether an arbitrary circuit is output-binding can be challenging, establishing this property for a *program* is significantly more straightforward. Specifically, a program represented as a sequence of instructions—executed in a step-by-step manner—is inherently output-binding. This observation motivates us to first represent CNN as a program defined over a public instruction set and then prove that the committed circuit is a correct compilation of some private program. Each advice value is only used for checking the execution of an individual instruction, making it easier to constrain the scope of their effect.

The next problem is that we need to ensure the program can be compiled into pR1CS with minimal overhead. In this work, we devise 4 types of *instructions*, including AddMult, Lookup, Div, MatMult, over a pR1CS-compatible computational model. CNN inference with different architectures can be expressed by this instruction set. Prover only need to prove the circuit is correctly compiled from a valid program. Detailed definition of the computational model and instruction set will be introduced in Section 5.

4. Parametrized Rank-one Constraint System

In this section, we present parametrized R1CS and its proof system generalized from R1CS and Spartan [4].

4.1. Parametrized R1CS

Vector $\vec{z}$ in conventional R1CS includes 2 parts: $\mathbb{x}$, referring to public I/O, and $\mathbb{w}$, referring to other gate values. In order to prove satisfiability of $\mathcal{C}(\mathbb{x})$, prover commits $\mathbb{w}$, and proves Eq. 1, where $\vec{z} = (1||\mathbb{x}||\mathbb{w})$. Mirage [43] incorporates randomized checking into R1CS and separates $\mathbb{w} = \mathbb{t}||\mathbb{a}$ into 2 parts, one that do not depend on the randomness and ones that do.

In pR1CS, a commitment to predetermined input, referring to model parameters, should be published in advance. Thus, $\vec{z}$ in pR1CS is separated into four parts, each of which is committed at different phases or in different manners:

- $\mathbb{p}$ refers to predetermined inputs, which correspond to the ML parameters. It is committed during the preprocessing phase, as the prover must ensure that identical parameters are used for all instances.
- $\mathbb{x}$ refers to public I/O from the verifier, corresponding to public inputs and outputs. It is completely public for the verifier.
- $\mathbb{t}$ refers to trace values in the intermediate arithmetic circuit, including all matrix multiplication I/O. It's committed before generating random challenge γ.
- $\mathbb{a}$ refers to auxiliary values or intermediate values that relevant to γ. It is committed after random challenge $\gamma \leftarrow_{\$} \mathbb{F}$ has been generated.

Namely, $\vec{z}$ is defined to be $(1||\mathbb{p}||\mathbb{x}||\mathbb{t}||\mathbb{a})$ in pR1CS.

The major difference of pR1CS compared with R1CS (and Mirage) is the inclusion of variable Γ into R1CS matrices to reduce online cost for checking matrix multiplication. Formally, not considering lookup, the circuit is represented by parametrized sparse matrix $A(\Gamma)$, $B(\Gamma)$, $C(\Gamma)$, each of which can be condensed into 4 vectors $\vec{row}_M, \vec{col}_M, \vec{val}_M, \vec{idx}_M$, such as

$$M(\Gamma)[\vec{row}_M[k], \vec{col}_M[k]] = \vec{val}_M[k] \cdot \Gamma^{\vec{idx}_M[k]}$$

Elements at undefined positions are 0. After committing $\mathbb{t}$, prover substituted a verifier-sampled $\gamma \leftarrow_{\$} \mathbb{F}$ into the variable matrices, and checks

$$A_\gamma \vec{z} \circ B_\gamma \vec{z} = C_\gamma \vec{z}$$

where M_γ is evaluating γ of variable matrix $M(\Gamma)$, namely

$$M_\gamma[\vec{row}_M[k], \vec{col}_M[k]] = \vec{val}_M[k] \cdot \gamma^{\vec{idx}_M[k]}$$

The formal definition of pR1CS is presented below:

Definition 2. *A parametrized R1CS structure consists of size bounds m, n, N and three parametrized matrices $A(\Gamma), B(\Gamma), C(\Gamma) \in (\mathbb{F} \times \mathbb{N})^{m \times n}$ with at most $N = \Omega(\max(m, n))$ non-zero entries in total. A parametrized R1CS input consists of $\mathbb{x}$. A parametrized R1CS witness consists of $\mathbb{t}$, and a map $\mathbb{a}(\cdot) : \mathbb{F} \to \mathbb{F}^a$. For $\varepsilon \in [0, 1]$, a pR1CS instance $A(\Gamma), B(\Gamma), C(\Gamma), \mathbb{x}$ is ϵ-satisfied by $\mathbb{t}, \mathbb{a}(\cdot)$ if there exists $S \subseteq \mathbb{F}$, $\frac{|S|}{|\mathbb{F}|} \geq \varepsilon$,*

$$A_\gamma \vec{z} \circ B_\gamma \vec{z} = C_\gamma \vec{z}$$

for every $\gamma \in S$.

Definition 3. *A proof system of pR1CS satisfies: For any pR1CS instance $A(\Gamma), B(\Gamma), C(\Gamma), \mathbb{x}$,*

- *if it's 1-satisfied by $\mathbb{t}, \mathbb{a}(\cdot)$, then the verification always pass (**completeness**).*
- *if it's $\frac{O(n)}{|\mathbb{F}|}$-satisfied by $\mathbb{t}, \mathbb{a}(\cdot)$, then for any probabilistic polynomial time adversarial prover, the verification pass with negligible probability (**soundness**).*

Using pR1CS, constraining a matrix multiplication only requires $n+1$ constraints, namely, computing each element of $(X^T\vec{\gamma}_1) \circ (Y\vec{\gamma}_2)$. The i-th multiplication gate computes

$$p[i] = \langle \vec{\gamma}_1, X^T[i] \rangle \cdot \langle \vec{\gamma}_2, Y[i] \rangle$$

and an additional one constraining

$$\sum_{i\in[n]} p[i] = \sum_{i\in[m],j\in[k]} \gamma^{ik+j} \cdot Z[i,j]$$

Lookup gates. Parametrized R1CS with lookup gates, includes two additional sparse matrices D and E, as well as a vector $\vec{tp}$, imposing the constraint $(D\vec{z}, E\vec{z}, \vec{tp}) \subseteq T$, where T is a public table with three columns. The vector $\vec{tp}$ is used to indicate the type of lookup. In the context of CNNs, the primary lookup operations are ReLU and GE0, which correspond to the ReLU activation and a table containing all values greater than or equal to zero, respectively.

4.2. Proof System

Proving the above constraint system is quite similar to Spartan, the state-of-the-art scheme for proving conventional R1CS. Its high-level idea is for a random vector $\vec{r}$, proving the following sumcheck:

$$\sum_{\vec{x}} \tilde{eq}_{\vec{r}}(\vec{x}) \cdot \left(\sum_{\vec{y}} \tilde{A}_\gamma(\vec{x},\vec{y}) \cdot \tilde{z}(\vec{y}) \right) \cdot \left(\sum_{\vec{y}} \tilde{B}_\gamma(\vec{x},\vec{y}) \cdot \tilde{z}(\vec{y}) \right)$$
$$\stackrel{?}{=} \sum_{\vec{x}} \tilde{eq}_{\vec{r}}(\vec{x}) \cdot \sum_{\vec{y}} \tilde{C}_\gamma(\vec{x},\vec{y}) \cdot \tilde{z}(\vec{y}) = \sum_{\vec{y}} \tilde{C}_\gamma(\vec{r},\vec{y}) \cdot \tilde{z}(\vec{y})$$

There are two differences in our generalized R1CS. 1) $\vec{z}$ is committed in multiple phases 2) matrices A, B, C has another term $\gamma^{i\vec{dx}[i]}$. The first difference can be solved by a batch opening of PCS. The second difference can be solved by adapting SPARK [4], the sparse PCS in Spartan. Specifically, proving $A_\gamma(\vec{x},\vec{y})$ can be translated into proving

$$\sum_k \vec{eq}_{\vec{x}}[r\vec{ow}[k]] \cdot \vec{eq}_{\vec{y}}\left[\vec{col}[k]\right] \cdot \vec{val}[k] \cdot \vec{\gamma}\left[\vec{idx}[k]\right] \quad (7)$$

where $\vec{\gamma} = (1, \gamma, \gamma^2, \cdots, \gamma^{n-1}) = \bigotimes_{i=0}^{\mu-1}(1, \gamma^{2^i})$ can be derived through tensor products. Thus, Equation 7 can be proved through an indexed lookup of these four vectors and conducting a sumcheck.

Amortized proof generation. If multiple instances have a shared random challenge, including $\vec{r}$ and γ, then their claimed evaluation of these sparse matrices are identical. Considering that the evaluation of these sparse matrices is the most costly part for the prover, amortizing them can help significantly reduce the prover cost. To aggregate challenges of multiple instances, prover can use the scheme in [44], merging challenges through a Merkle tree and use the root as challenge next round. We present the full amortized proof generation procedure of our scheme in Protocol 3, detailed in Appendix B.

Theorem 1. *Protocol 3 is a PIOP of parametrized R1CS with soundness error $\frac{O(n)}{|\mathbb{F}|}$.*

4.3. Adding Zero-knowledge.

There has been extensive work [2], [3] on adding zero-knowledge to sumcheck-based SNARKs through committing a masked polynomial. The additional cost of incorporating zero-knowledge is an $O(1)$ increase in proof size and an $O(\log n)$ increase in prover time.

However, unlike conventional zkSNARKs where polynomials are only opened at a few points, zkML requires substantial openings of the polynomial $\mathbb{p}$. As a result, any one-time masking becomes ineffective, since evaluating the polynomial $O(|\mathbb{p}|)$ times will fully reveal its contents. This problem can be addressed by re-masking the polynomial for each proof. Specifically, the prover can leverage the homomorphic properties of PCS by adding a term $R(X_\mu) = r(X_\mu) \cdot X_\mu(1 - X_\mu)$ to the masked polynomial $\tilde{\mathbb{p}}$, where $r(X_\mu)$ is a low-degree random polynomial. The prover must additionally prove that $R(0) = R(1) = 0$. This masking polynomial guarantees that it does not affect the evaluations over the hypercube. With this technique, each proof contains evaluations of different masked polynomials, preventing the verifier from inferring model parameters through multiple proofs.

Our architecture-private zkML setting is more complex, as it aims to hide the committed circuit, which involves re-masking a *sparse* polynomial. This issue can also be addressed by applying the same masking technique: masking the matrix M with a low-degree polynomial that evaluates to zero on the entire hypercube. The main challenge is that the sparse polynomial $M(\vec{X},\vec{Y})$ is not directly committed via PCS, making it difficult to mask directly. To overcome this, we introduce random terms by appending elements to the committed vectors and extending the sparse matrix with additional rows and columns.

Suppose we want to add two non-zero terms $r_1 \cdot \vec{X}[-1](1 - \vec{X}[-1])$ and $r_2 \cdot \vec{Y}[-1](1 - \vec{Y}[-1])$ to sparse polynomial $\tilde{A}(\vec{X},\vec{Y})$. We can add 2 terms to $\vec{eq}_{\vec{x}}$ and $\vec{eq}_{\vec{y}}$ when proving Equation 7. More precisely, assume sparse matrix A has N rows and M columns, we define $\vec{Z}_1$ of length $N+2$ as $\vec{Z}_1[i] = \vec{eq}_{\vec{x}}[i]$ for $i < N$, $\vec{Z}_1[N] = \vec{x}[-1]$, $\vec{Z}_1[N+1] = \vec{y}[-1]$. Define $\vec{Z}_2$ of length $M+2$ such that $\vec{Z}_2[i] = \vec{eq}_{\vec{y}}[i]$ for $i < M$, $\vec{Z}_2[M] = 1 - \vec{x}[-1]$, $\vec{Z}_2[M+1] = 1 - \vec{y}[-1]$. When proving $A_\gamma(\vec{x},\vec{y})$, prover alternatively proves

$$\sum_k \vec{Z}_1[r\vec{ow}[k]] \cdot \vec{Z}_2\left[\vec{col}[k]\right] \cdot \vec{val}[k] \cdot \vec{\gamma}\left[\vec{idx}[k]\right]$$

Protocol 1. ZK sparse polynomial commitment

Sparse matrix $M(\Gamma)$ can be parsed into 4 vectors $\vec{row}, \vec{col}, \vec{val}, \vec{idx}$ of length n. Assume the matrix size is $N \times M$.

- Commit: Append $(N, N+1)$ to $\vec{row}$. Append $(M, M+1)$ to $\vec{col}$. Append $(0,0)$ to $\vec{val}$ and $\vec{idx}$ respectively. Prover invokes PC.Commit to commit $\tilde{row}$, $\tilde{col}$, $\tilde{val}$, $\tilde{idx}$.
- Remask: Commit random polynomial $\tilde{R}(\vec{X}) = r_1 \cdot \tilde{eq}_{\mathsf{bin}(n)}(\vec{X}) + r_2 \cdot \tilde{eq}_{\mathsf{bin}(n+1)}(\vec{X})$, where $r_1, r_2 \leftarrow_{\$} \mathbb{F}$ are two uniform random scalars. Perform a zero-knowledge sumcheck to prove $\tilde{R}(\vec{X})$ evaluates 0 on hypercube except $\mathsf{bin}(n)$ and $\mathsf{bin}(n+1)$. Set $\tilde{val} \leftarrow \tilde{val} + \tilde{R}$.
- Open: To open $\tilde{M}_\gamma(\vec{z}_1, \vec{z}_2) = y$, prover and verifier perform the following
  1) Define $\vec{\gamma} := (\gamma^0, \gamma^1, \cdots, \gamma^{2^\mu - 1})$. Define $\vec{Z}_1$ such that $\vec{Z}_1[i] = \vec{eq}_{\vec{z}_1}[i]$ for $i < N$, $\vec{Z}_1[N] = \vec{z}_1[-1]$, $\vec{Z}_1[N+1] = \vec{z}_2[-1]$. Define $\vec{Z}_2$ such that $\vec{Z}_2[i] = \vec{eq}_{\vec{z}_2}[i]$ for $i < M$, $\vec{Z}_2[M] = 1 - \vec{z}_1[-1]$, $\vec{Z}_2[M+1] = 1 - \vec{z}_2[-1]$.
  2) Prover invokes PC.Commit to commit val_r, $\tilde{val}_c$, $\tilde{val}_i$, where $\vec{val}_r[\cdot] := \vec{Z}_1[\vec{row}[\cdot]]$, $\vec{val}_c[\cdot] := \vec{Z}_2[\vec{col}[\cdot]]$, $\vec{val}_i := \vec{\gamma}[\vec{idx}[\cdot]]$.
  3) Prover and verifier invoke an indexed lookup to prove the correctness of these three committed vectors. It would finally reduce to random evaluations of $\tilde{val}_r$, val_c, $\tilde{val}_i$, $\tilde{row}$, $\tilde{col}$, $\tilde{val}$, which can be proved through invoking PC.Eval, and random evaluations of $\tilde{Z}_1$, $\tilde{Z}_2$ and $\tilde{\gamma}$, which can be evaluated efficiently by verifier.
  4) Prover and verifier invokes a sumcheck to prove
  $$\sum_{i \in [n+2]} \left(\vec{val}_r \circ \vec{val}_c \circ \vec{val}_i \circ \vec{val} \right)[i] = y$$
  5) Assume the randomness of sumcheck is $\vec{r}$, prover invokes PC.Eval to prove the evaluation of $\tilde{val}_r(\vec{r})$, $\tilde{val}_c(\vec{r})$, $\tilde{val}_i(\vec{r})$, $\tilde{val}(\vec{r})$.

The re-masking of sparse polynomial can be changing the value in $\vec{val}$ corresponding to last two rows and columns. Detailed protocol is presented in Protocol 1.

Theorem 2. *Protocol 1 is a ZK-PIOP for sparse matrix evaluation, achieving a soundness error of $\frac{O(n)}{|\mathbb{F}|}$ regardless of the number of re-maskings and openings performed.*

5. Architecture-private ZKP for CNN

In this section, we demonstrate how to construct architecture-private ZKP for CNN based on pR1CS. In our construction, prover first commits the inference function as a *program* that sequentially runs *instructions* over a public instruction set. Prover only needs to demonstrate that the committed program (1) is valid, and (2) has been *correctly compiled* into the committed private circuit. In Section 5.1, we present our pR1CS-compatible virtual machine and its instruction set. In Section 5.2, we introduce the method to commit a private program. In Section 5.3, we present how to check validity of the program and prove the pR1CS circuit is correctly compiled from the committed program.

5.1. Virtual Machine and Instructions

Our virtual machine (VM) is highly compatible with pR1CS. A *program* of our VM is described by pR1CS matrices $A(\Gamma)$, $B(\Gamma)$, D, $\vec{tp}$ and a sequence of instructions. They can be understood as static values and instructions in conventional X86 executable files. The VM works as follows: Initially, the trace $\mathbb{t}$ is empty. Each instruction will read elements on input $\mathbb{x}$ and current trace $\mathbb{t}$ at specific positions, and append the corresponding output to the end of $\mathbb{t}$. It may also write some auxiliary output to $\mathbb{a}$ used for verification.

We devise 4 types of instructions that are able to fully implement all architectures of CNN. Each instruction has an operation name and several operation numbers. Assuming the program is currently writing to $\vec{z}[t] \in \mathbb{t}$, syntax, execution functionality, and checking methods of instructions are presented as follows:

- $[\mathsf{AddMult}, cr]$: output trace value $\vec{z}[t] = \langle A[cr], \vec{z} \rangle \cdot \langle B[cr], \vec{z} \rangle$. It's checked by R1CS constraints itself.
- $[\mathsf{Lookup}, lr]$: Output trace value $\vec{z}[t] = T_{\vec{tp}[lr]}(\langle D[lr], \vec{z} \rangle)$, which means using $\langle D[lr], \vec{z} \rangle$ as input to lookup the output in table $T_{\vec{tp}[lr]}$. It can be checked by lookup arguments directly. It is used for both the activation function and max-pooling (noticing $\max(a,b) = a + \mathsf{ReLU}(b-a)$).
- $[\mathsf{Div}, d, cr, lr, aux]$: Define $D = \langle A[cr], \vec{z} \rangle \cdot \langle B[cr], \vec{z} \rangle$. Output
  1) Trace value $\vec{z}[t] = \lfloor \frac{D}{d} \rfloor$
  2) Auxiliary value $\vec{z}[aux] = D - \vec{z}[t] \cdot d$
  3) Auxiliary value $\vec{z}[aux+1] = \vec{z}[aux] \cdot (d - 1 - \vec{z}[aux])$

  Checking the correctness of $\vec{z}[t]$ can be achieved by checking
  1) $D = \vec{z}[aux] + d \cdot \vec{z}[t]$,
  2) $\vec{z}[aux+1] = \vec{z}[aux] \cdot (d - 1 - \vec{z}[aux])$ in the cr-th and $(cr+1)$-th row of R1CS constraints,
  3) $\vec{z}[aux+1] \geq 0$, which is checked in the lr-th row of R1CS lookup.

  Div is used for both quantization and average pooling.
- $[\mathsf{MatMult}, m, n, k, s_1, s_2, s_p]$: Define matrix $X \in \mathbb{F}^{m \times n}$,

$Y \in \mathbb{F}^{n\times k}$, where $X[i,j] = \vec{z}[s_1 + i + jm]$, $Y[i,j] = \vec{z}[s_2 + ik + j]$. It outputs

1) A flattened matrix $Z = X \cdot Y$ in trace, where $\vec{z}[t + ik + j] = Z[i,j]$
2) A vector $\vec{p}$ of length n in auxiliary, where $\vec{z}[s_p + i] = \vec{p}[i] = \langle \vec{\gamma}_1, X^T[i] \rangle \cdot \langle \vec{\gamma}_2, Y[i] \rangle$

It is used for both convolutional layers as mentioned in Section 2.6, and fully connected layers.

We illustrate the execution of the virtual machine with an example program, say $[[\mathsf{AddMult}, cr = 0], [\mathsf{Div}, cr = 1, d = 4, lr = 0, aux = 6]]$, and matrices

$$A = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Assume the input is $(1,2,3)$, then initially, we have $\vec{z} = (1,2,3,0,0,0,0)$ On the AddMult instruction, we compute $\langle A[0], \vec{z} \rangle \cdot \langle B[0], \vec{z} \rangle = (1-2) \cdot 6 = -6$ and we append -6 to trace part of $\vec{z}$ to derive

$$\vec{z} = (1,2,3,-6,0,0,0)$$

On the Div instruction, we compute $\left\lfloor \frac{\langle A[1], \vec{z} \rangle \cdot \langle B[1], \vec{z} \rangle}{4} \right\rfloor = \left\lfloor \frac{5 \cdot (2+3)}{4} \right\rfloor = 6$, and we append 6 to trace part of $\vec{z}$. Moreover, this instruction will write auxiliary values $25 - 6 \times 4 = 1$ to $\vec{z}[6]$, and $1 \times ((4-1)-1) = 2$ to $\vec{z}[7]$. At the end of this small program, we have

$$\vec{z} = (1,2,3,-6,6,0,1,2)$$

where $\mathbb{t} = (-6,6)$ and $\mathbb{a} = (1,2)$.

Validity of program. Note that not all arbitrary A, B, D forms a valid program. In this example, all non-zero columns of $A[0]$, $B[0]$ should be smaller than 3, and the threshold of $A[1]$, $B[1]$ should be 4. Since $A[1]$, $B[1]$ are used for inputs to Div, according to its checker, we need to check $A[2]$, $B[2]$ forms the computation for $\vec{z}[6](d-1-\vec{z}[6])$, and $D[0]$, $\vec{tp}[0]$ forms the lookup for $\vec{z}[7] \in \mathsf{GE0}$. We will introduce how to check validity of the program in Section 5.3.

Nonetheless, for any valid program comprised of our instruction set, the entire trace must be determined by input $\mathbb{x}$, so the program is output-binding.

5.2. Commitment of private program

In addition to committing the pR1CS tuples, prover commits the private program by committing selectors $\left(\vec{sel}_i\right)_{i=1}^{4} \in \{0,1\}^{n\times 4}$ and operation numbers $(\vec{opn}_i)_{i=1}^{4} \in \mathbb{F}^{n\times 4}$, where $n = |\mathbb{t}|$.

Instruction	Operation Numbers			
AddMult	cr			
Lookup	lr			
Div	d	cr	lr	aux
MatMult	s_o	m	k	

TABLE 1: Operation number table for each instruction

Selectors are used to mark which instruction is used to derive the corresponding *trace* value. For example, if $\mathbb{t}[i]$ is derived through Div, then there should be

$$\vec{sel}_j[i] = \begin{cases} 0 & j \neq 3 \\ 1 & j = 3 \end{cases}$$

After committing these 4 selectors, prover needs to prove that:

$$\forall j \in [1,4], \vec{sel}_j \circ (\vec{sel}_j - \vec{1}) = \vec{0} \wedge \sum_{j=1}^{4} \vec{sel}_j = \vec{1}$$

Operation number of the i-th instruction is provided at $\vec{opn}_j[i]$, $j \in [1,4]$. How to parse these 4 operation numbers is presented in Table 1, where blank positions will not be used. For example, if $\mathbb{t}[i]$ is derived through lookup, then $\vec{opn}_1[i]$ means its lookup row, while $\vec{opn}_2[i]$ can be any value as it will not be used afterwards.

5.3. Proving Correct Compilation

In this section, we formally present how to check

- the committed program is valid, primarily involving matrix $A(\Gamma)$, $B(\Gamma)$, D
- the committed circuit is correctly compiled from the program, primarily involving matrix $C(\Gamma)$, E, $\vec{tp}$

After compilation, the constraints must be suffice to limit $\mathbb{t}$ adheres to the execution result of the committed program.

Before presenting constraints of each instruction, we first introduce some useful tools to constraint the rows in sparse matrices. The prover needs to commit extra vectors $\vec{num}_M$ for $M \in \{A,B,C,D,E\}$ representing the number of non-zero columns in each row, and $\vec{max}_M$ for $M \in \{A,B,D\}$ representing the threshold index of non-zero column in each row respectively. The prover can prove $\vec{num}_M$ by proving $M'(\vec{r},\vec{1}) = \vec{num}_M(\vec{r})$, where M' is defined by setting all non-zero elements in M to 1, and $\vec{r}$ is a random vector. $\vec{max}_M$ can be proved by another indexed lookup vector $\vec{mc}[k] = \vec{max}[\vec{row}[k]]$ satisfying that $\vec{mc}[k] > \vec{col}[k]$ for each k.

Next, we introduce how to leverage the tools mentioned above, including selectors, operation numbers, $\vec{num}_M$ and $\vec{max}_M$ to check that the whole circuit is honestly compiled from a valid program. We will discuss constraints for instructions one by one. For simplicity, we define $\vec{t}$ as interval vector $[1 + |\mathbb{p}| + |\mathbb{x}|, 1 + |\mathbb{p}| + |\mathbb{x}| + |\mathbb{t}|)$.

5.3.1. AddMult. Assuming the $\vec{z}[i]$ is derived through AddMult, verifier needs to check:

- all non-zero columns in $A[cr]$, $B[cr]$ are smaller than i, translated into:

$$(o\vec{p}n_1, \vec{t}) \circ s\vec{e}l_1 \subseteq ([n], m\vec{a}x_A) \cup (0,0)$$
$$(o\vec{p}n_1, \vec{t}) \circ s\vec{e}l_1 \subseteq ([n], m\vec{a}x_B) \cup (0,0)$$

- $C[cr] = \{(i,1)\}$, translated into:

$$(o\vec{p}n_1, \vec{1}) \circ s\vec{e}l_1 \subseteq ([n], n\vec{u}m_C) \cup (0,0)$$
$$(o\vec{p}n_1, \vec{t}, \vec{1}, \vec{0}) \circ s\vec{e}l_1 \subseteq (r\vec{o}w_C, c\vec{o}l_C, v\vec{a}l_C, i\vec{d}x_C) \cup (0,0,0,0)$$

Note that we don't need to check the $i\vec{d}x$ of non-zero elements in $A[cr]$ and $B[cr]$, because if either of $i\vec{d}x$ is non-zero, then by the Schwartz-Zippel Lemma, for a randomly chosen $\gamma \leftarrow_{\$} \mathbb{F}$, the probability that $\langle A_\gamma, \vec{z}\rangle \cdot \langle B_\gamma, \vec{z}\rangle = \vec{z}[i]$ holds is negligible.

5.3.2. Lookup and Div. Due to page limit, we put the constraints for Lookup and Div in Appendix C, as they can be checked through very similar ideas.

5.3.3. MatMult. Matrix multiplication is significantly more complex than other instructions. One of the key differences is that MatMult produces outputs of variable length. Our key insight to address this challenge is to verify the existence of a variable number of consecutive constraints by checking: 1) the first and last constraints exist 2) each constraint is either the initial one or an increment over the previous constraint.

We introduce another table MM to record all matrix multiplications, so checking MatMult instruction constraints only requires checking:

- the matrix multiplication has been recorded in MM, translated into

$$(o\vec{p}n_1, o\vec{p}n_2, o\vec{p}n_3) \circ s\vec{e}l_4 \subseteq (\mathsf{MM}.\vec{s}_o, \mathsf{MM}.\vec{M}, \mathsf{MM}.\vec{K})$$

- $\vec{z}[i]$ belongs to the output matrix, translated into

$$(\vec{t} - o\vec{p}n_1) \circ s\vec{e}l_4 \subseteq \mathsf{GE0}$$
$$(o\vec{p}n_1 + o\vec{p}n_2 \circ o\vec{p}n_3 - \vec{t}) \circ s\vec{e}l_4 \subseteq \mathsf{GE0}$$

Details about table MM and how to check that all recorded matrix multiplications are finely constrained will be presented in Appendix C.

5.4. Putting everything together

We put everything together and present a formal description of our proof of functional relation scheme in Protocol 2.

Theorem 3 (PFR Soundness)**.** *Define $\mathcal{F}$ as the set of all functions representable by our instruction set, with $\vec{z} = 1||\mathbb{x}||\mathbb{w}||\mathbb{t}||\mathbb{a}$ such that $|\mathbb{x}|, |\mathbb{w}|, |\mathbb{t}|, |\mathbb{a}|$ are all public values. Our solution can prove that the committed circuits corresponds to a function $f \in \mathcal{F}$.*

Theorem 4 (Hiding)**.** *When commitments of sparse matrices $A(\Gamma)$, $B(\Gamma)$, $C(\Gamma)$, D, E and vector $\vec{tp}$ uses a PCS with hiding property, the proof of functional relation scheme (PFR) and the proof system satisfies zero-knowledge, then the whole protocol is function-hiding.*

Proofs for Theorem 3 and Theorem 4 are deferred to Appendix D.

Corollary 1. *Combining Theorem 1, Theorem 3 and Theorem 4, the whole protocol is a function-hiding functional commitment scheme.*

Proof. Due to the binding property of the PCS, our commitment to the pR1CS circuit is obviously also binding. The soundness has been implied by Theorem 1 and Theorem 3, while the hiding is implied by Theorem 4. □

This implies that our entire protocol constitutes an architecture-private ZKP scheme for CNN. We note that although the committed function does not necessarily have to be a neural network, it must be output-binding. This requirement ensures both integrity and fairness for the ML model owner, which aligns with the core functionality desired in zkML.

6. Evaluation

In this section, we present evaluations of our proposed schemes from both theoretical and empirical analysis.

6.1. Theoretical Analysis

Proof system. When proving a single instance, the prover cost for a pR1CS circuit of size n is $O(n)$, while the communication overhead and verifier time are both $O(\log n)$. This matches the performance of Spartan and other state-of-the-art linear-time proof systems. When amortizing proofs across N users, the verifier time and communication cost increase to $O(\log N \log n)$, as each round of interaction requires an $O(\log N)$ Merkle path.

When instantiated with a CNN, the circuit size remains asymptotically equivalent to that of BFG+23 [13], resulting in the same asymptotic prover complexity. Compared to zkCNN [7], our approach benefits from BFG+23's more efficient compilation from CNNs to arithmetic circuits, yielding improved asymptotic prover performance. The proof size of our solution can be asymptotically better than BFG+23, as the GKR-based solution has proof size linear to layer number d.

Proof of functional relation. The cost of proving functional correctness is also linear with regard to the circuit size. Verifier time and proof size are $O(\log n)$, where n is the circuit size. Since this is performed only once during a preprocessing phase, the concrete cost is not a significant concern.

6.2. Experimental Evaluation

Software. We implemented the proof system for pR1CS in Rust and used it to construct a SNARK for CNN model,

Protocol 2. Proof of functional relation scheme

- Prover input: program P; Circuit compiled from program P, i.e. sparse matrices $A(\Gamma)$, $B(\Gamma)$, $C(\Gamma)$, D, E and vector $\vec{tp}$.
- Public input: Commitments of sparse matrices $A(\Gamma)$, $B(\Gamma)$, $C(\Gamma)$, D, E, $\tilde{tp}$.
- Prove: $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, \mathcal{C}, \mathbb{w}, \{\mathbb{x}_i\}_{i\in[N]})$
  1) Prover commits vector tuples $\left(\vec{sel}_i\right)$ of length $|\mathbb{t}|$, representing how each trace value is derived. Prover proves $\vec{sel}_j \circ (\vec{sel}_j - \vec{1}) = \vec{0}$ and $\sum \vec{sel}_j = \vec{1}$.
  2) Prover commits vector tuples $(\vec{opn}_i)$ of length $|\mathbb{t}|$. Undefined positions can be padded with random numbers.
  3) For $M \in \{A, B, D\}$, prover commits $\vec{max}_M$, where $\vec{max}_M[i]$ represents the maximum non-zero limit of the $M[i]$. If $M[i]$ doesn't have limitation, set $\vec{max}_M[i]$ to be width of M; Prover invokes $\mathsf{PC.Commit}$ to commit $\tilde{mc}_M$, where $\vec{mc}_M[\cdot] = \vec{max}[\vec{row}[\cdot]]$, and proves $(\vec{row}_M, \vec{mc}_M) \subseteq ([n], \vec{max}_M)$. Prover proves $\vec{mc}_M - \vec{col}_M - \vec{1} \subseteq \mathsf{GE0}$.
  4) For $M \in \{A, B, C, D, E\}$, prover commits $\vec{num}_M$, where $\vec{num}_M[i]$ represents the number of non-zero elements in $M[i]$; Define sparse matrix M' to be setting all non-zero elements in M to be 1, i.e. dense representation of M' is $(\vec{row}_M, \vec{col}_M, \vec{1})$. Verifier samples $\vec{r} \in_R \mathbb{F}^\mu$, and prover proves $\tilde{M}_1(\vec{r}, \vec{1}) = \tilde{num}_M(\vec{r})$ by invoking Protocol 1 and $\mathsf{PC.Eval}$. This can ensure $M_1 \cdot \vec{1} = \vec{num}_M$.
  5) For each instruction, prover proves constraints presented in Section 5.3.
  6) Prover invokes $\mathsf{PC.Commit}$ to commit all columns of MM, Inprod, CstA, CstB, CstC tables. Prover proves constraints of these tables presented in Section C.

Schemes	VGG11			VGG16			Hide arch?
	Prover time	Verifier time	Communication	Prover time	Verifier time	Communication	
zkCNN [7]	51.6 s	243 ms	284KB	84.6 s	346 ms	316 KB	no
BFG+23 [13]	20.1 s	138 ms	150 KB	27.4s	179 ms	193 KB	no
Our single	288 s	23 ms	15 KB	390 s	24 ms	16 KB	yes
Our amortized	28.3 s	27 ms	42 KB	37.5s	28 ms	43 KB	yes

TABLE 2: Prover time of different zkSNARKSgiv over two CNN models

(m, n, k)	**pR1CS**	R1CS	Thaler 13
$1024, 576, 64$	1.81 s	5.52 s	1.40 s
$256, 576, 128$	0.54 s	1.7s	0.48 s
$256, 1152, 128$	1.09 s	3.38 s	0.97 s
$256, 768, 2304$	6.65 s	21.1 s	5.46 s
$1024, 768, 2304$	13.2 s	43.8 s	10.1s

TABLE 3: Online prover cost for matrix multiplication

achieving a security level of over 100 bits. For field arithmetic over the roughly 256-bit prime field of the BN254 curve, we utilize the Arkworks library [45]. Our scheme adopts layered multilinear KZG [46] as the polynomial commitment scheme to ensure low evaluation costs. Following prior work, we set the quantization scale hyperparameter to $2^Q = 256$.

Hardware. We carry out different benchmarks in a server running Ubuntu 22.04 LTS with an Intel Xeon 6126 2.6GHz 16-core CPU and 200GB of memory. All the experiments are run in single thread.

6.2.1. Architecture-private zkSNARK for CNN. We benchmark our end-to-end architecture-private zkSNARK for CNN, and compare the cost with existing architecture-public zkSNARKs schemes for CNN, demonstrating our technique of hiding architecture incurs acceptable prover overhead, especially in the context of amortized proving.

Dataset and Models. Following zkCNN [7], we evaluate our system using the popular CIFAR-10 dataset. It is a classification dataset taking image size $32 \times 32 \times 3$ as input. Following zkCNN [7] and BFG+23 [13], we evaluate two neural networks: VGG-11 and VGG-16.

Baselines. In this paper, we compare the concrete performance of our system with existing zkSNARK schemes for proving CNN inference. Our baselines include zkCNN (CCS'21) and BFG+23 (CCS'23). The open-source implementation of BFG+23 only supports proving a single layer, rather than an end-to-end CNN. We re-implement their end-to-end scheme ourselves, using the same components as our own system. For our system, we benchmark the performance of proving a single instance and the amortized performance when proving 64 instances concurrently, which is a normal batch size setting in ML deployment [47].

The benchmark result is shown in Table 2. When only proving a single instance, the prover cost of our scheme on VGG-16 model is 390s, about 4.6× slower than zkCNN, 14× slower than BFG+23. When amortized over proving 64 instances, our amortized proof generation cost is 37.5s, roughly 2.3× faster than zkCNN and roughly 30% slower than BFG+23. Our scheme offers significantly improved verifier time and proof size relative to zkCNN and BFG+23, as pR1CS avoids the linear dependence on the layer number d that arises in GKR. Although this optimization is not very important in MLaaS, this may be of independent interests for on-chain ML [48], [49].

How Much Can BFG+23 Benefit from Amortization? Since BFG+23 is designed to prove only a single prediction result, it might seem unfair to directly compare its performance with the amortized proof generation in our scheme. A natural question arises: can BFG+23 also benefit from amortization, similar to the improvements observed in our approach? To explore this, we provide a theoretical analysis to estimate the potential benefits of amortization for BFG+23. The total runtime of BFG+23 is approximately 27.4 seconds, which can be roughly decomposed into four main components: (1) PCS witness commitment — 3s, (2) sumcheck — 7s, (3) PCS opening (including both weights and witness) — 3s, and (4) Logup — 14s. Among these, the costs of Logup and PCS commitment are difficult to amortize due to their strong dependence on specific instances. Therefore, based on this coarse-grained analysis, amortization could reduce at most around 30% of BFG+23's total cost, in contrast to our construction, which achieves more than $10\times$ improvement through amortization.

Proof of functional relation. Since the proof of the functional relation is executed only once during a preprocessing phase, its exact execution time is not critical. Therefore, we did not implement this step and leave it as potential future work. Based on a conservative estimate, the overall runtime of the PFR scheme for VGG16 is less than one hour, as the prover's time scales linearly with the circuit size.

6.2.2. Matrix multiplication in pR1CS. In this section, we evaluate the benefits of our proposed pR1CS construction over conventional R1CS. Since matrix multiplication serves as a core primitive in zkML, we focus on the *online cost* of generating proofs for matrix multiplication. We compare three schemes: (i) our pR1CS, (ii) standard R1CS, and (iii) Thaler13 [40], the most widely used approach for verifiable matrix multiplication in architecture-public zkML. For fairness, PCS of all these scheme are instantiated with layer-multilinear KZG [46].

Experiment setting. We benchmark verifiable matrix multiplications $X \cdot Y = Z$ for different sizes, where $X \in \mathbb{F}^{m\times n}$, $Y \in \mathbb{F}^{n\times k}$, and $Z \in \mathbb{F}^{m\times k}$. The benchmarks are conducted for the following configurations of (m, n, k): $(1024, 576, 64)$, $(256, 576, 128)$, and $(256, 1152, 128)$, which correspond to matrix multiplications used in VGG-16; and $(256, 768, 2304)$ and $(1024, 768, 2304)$, which are used in GPT-2. In particular, the 768×2304 matrix corresponds to the attention matrix in GPT-2. For the input matrices, we set X and Y to be random matrices where each element has an absolute value smaller than 2^{20}. This is because, even after quantization, the values of both neuron outputs and weights are not expected to be significantly large.

The experiment results are presented in Table 3. We find that pR1CS is roughly 3-4$\times$ faster than conventional R1CS on average. The advantage comes from the fact that $\vec{z}$ to commit in pR1CS is only around half the length as in conventional R1CS. Moreover, all the scalars in pR1CS's $\vec{z}$ are smaller than 2^{40}, while R1CS has many large field elements. When using elliptic curve-based schemes, committing smaller field elements are much cheaper [38].

Compared to Thaler13 [40], pR1CS incurs only about 30% higher cost. This is because Thaler13's overhead is primarily due to committing to the matrices via a PCS and opening them once. In contrast, pR1CS requires more sumcheck invocations than Thaler13. The key advantage of pR1CS is its ability to prove matrix multiplications of varying sizes across different layers within a single proof, while obfuscating both the individual dimensions and the total number of multiplications. Given that Thaler13 is the most efficient solution to prove matrix multiplication, we believe that pR1CS achieves nearly optimal performance for concurrently proving multiple matrix multiplications without revealing their specific sizes and multiplication counts.

6.3. Discussion of Prover Efficiency

A limitation of our proposed scheme is that it is less efficient than existing public-architecture zkML approach. However, as the first work to achieve architecture-private. this trade-off in efficiency is acceptable. Achieving the same level of efficiency as public-architecture zkML remains an important direction for future work.

Moreover, in many practical scenarios, security can be more critical than efficiency, as model server doesn't have to attach a proof for *every* inference result. In practice, inference results can be stored, and a ZKP proof can be generated only when the result is questioned. For instance, in AI-assisted medicine [50], [51], the model server would need to provide a proof only if its output has led to a dispute or adverse outcome. This mechanism can effectively regulate model servers, preventing their use of biased or simplified models. In such cases, only a small fraction of inferences would require proof generation, greatly mitigating the efficiency concern.

7. Conclusion

In this work, we initiate the study of *architecture-private* zero-knowledge proofs for CNNs. Our approach ensures the integrity of the model server without revealing either the model architecture or the model weights. The high-level idea is to have the model server publish a commitment to the model architecture and then prove that the inference result was computed using this committed model. The core technique of our scheme is pR1CS, a constraint system specifically designed for efficient matrix multiplication checking. Another key component is the design of a pR1CS-compatible virtual machine, which enables proving that the committed circuit corresponds to an output-binding function. Experimental results show that when batch-proving 64 instances, the prover is only 30-40% slower than the previous state-of-the-art CNN proving scheme (CCS'23).

8. Ethics Considerations

None.

9. LLM usage considerations

LLMs were used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality.

References

[1] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019. [Online]. Available: https://eprint.iacr.org/2019/953

[2] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 733–764.

[3] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "Hyperplonk: Plonk with linear-time prover and high-degree custom gates," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 499–530.

[4] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[5] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "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*. Springer, 2020, pp. 738–768.

[6] J. Zhang, Z. Fang, Y. Zhang, and D. Song, "Zero knowledge proofs for decision tree predictions and accuracy," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 2039–2053.

[7] T. Liu, X. Xie, and Y. Zhang, "zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985.

[8] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4405–4419.

[9] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[10] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[11] M. Campanelli, A. Faonio, D. Fiore, T. Li, and H. Lipmaa, "Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees," in *IACR International Conference on Public-Key Cryptography*. Springer, 2024, pp. 337–369.

[12] S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network based on zk-snarks," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 4254–4270, 2024.

[13] D. Balbás, D. Fiore, M. I. González Vasco, D. Robissout, and C. Soriente, "Modular sumcheck proofs with applications to machine learning and image processing," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1437–1451.

[14] M. Hao, H. Chen, H. Li, C. Weng, Y. Zhang, H. Yang, and T. Zhang, "Scalable zero-knowledge proofs for non-linear functions in machine learning," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3819–3836.

[15] W. Qu, Y. Sun, X. Liu, T. Lu, Y. Guo, K. Chen, and J. Zhang, "zkgpt: An efficient non-interactive zero-knowledge proof framework for llm inference," in *34st USENIX Security Symposium (USENIX Security 25)*, 2025.

[16] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[17] "Openai is growing fast and burning through piles of money," 2024. [Online]. Available: https://www.nytimes.com/2024/09/27/technology/openai-chatgpt-investors-funding.html

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "Zkml: An optimizing system for ml inference in zero-knowledge proofs," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 560–574.

[20] D. Boneh, W. Nguyen, and A. Ozdemir, "Efficient functional commitments: How to commit to a private function," *Cryptology ePrint Archive*, 2021.

[21] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 501–518.

[22] B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences," *Cryptology ePrint Archive*, 2021.

[23] B. Libert, S. C. Ramanna, and M. Yung, "Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions," in *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, 2016.

[24] H. Lipmaa and K. Pavlyk, "Succinct functional commitment for a large class of arithmetic circuits," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 686–716.

[25] H. Wee and D. J. Wu, "Succinct vector, polynomial, and functional commitments from lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 385–416.

[26] ——, "Succinct functional commitments for circuits from k-lin," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 280–310.

[27] L. de Castro and C. Peikert, "Functional commitments for all functions, with transparent setup and from sis," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 287–320.

[28] R. Ghosal, A. Sahai, and B. Waters, "Non-interactive publicly-verifiable delegation of committed programs," in *IACR International Conference on Public-Key Cryptography*. Springer, 2023, pp. 575–605.

[29] S. Setty, J. Thaler, and R. Wahby, "Customizable constraint systems for succinct arguments," *Cryptology ePrint Archive*, 2023.

[30] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Theory of Cryptography Conference*. Springer, 2013, pp. 222–242.

[31] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 926–943.

[32] H. Zeilberger, B. Chen, and B. Fisch, "Basefold: efficient field-agnostic polynomial commitment schemes from foldable codes," in *Annual International Cryptology Conference*. Springer, 2024, pp. 138–169.

[33] Y. Guo, X. Liu, K. Huang, W. Qu, T. Tao, and J. Zhang, "Deepfold: Efficient multilinear polynomial commitment from reed-solomon code and its application to zero-knowledge proofs," *Cryptology ePrint Archive*, 2024.

[34] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," *Cryptology ePrint Archive*, 2020.

[35] U. Haböck, "Multivariate lookups based on logarithmic derivatives," *Cryptology ePrint Archive*, 2022.

[36] L. Eagen and U. Haböck, "Bypassing the characteristic bound in logup," *Cryptology ePrint Archive*, 2024.

[37] S. Papini and U. Haböck, "Improving logarithmic derivative lookups using gkr," *Cryptology ePrint Archive*, 2023.

[38] S. Setty, J. Thaler, and R. Wahby, "Unlocking the lookup singularity with lasso," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 180–209.

[39] R. Freivalds, "Probabilistic machines can use less running time." in *IFIP congress*, vol. 839, 1977, p. 842.

[40] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Annual Cryptology Conference*. Springer, 2013, pp. 71–89.

[41] M. Campanelli, D. Fiore, and R. Gennaro, "Natively compatible super-efficient lookup arguments and how to apply them," *Journal of Cryptology*, vol. 38, no. 1, p. 14, 2025.

[42] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vsql: Verifying arbitrary sql queries over dynamic outsourced databases," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 863–880.

[43] A. Kosba, D. Papadopoulos, C. Papamanthou, and D. Song, "{MIRAGE}: Succinct arguments for randomized algorithms with applications to universal {zk-SNARKs}," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2129–2146.

[44] J. Zhang, T. Xie, T. Hoang, E. Shi, and Y. Zhang, "Polynomial commitment with a {One-to-Many} prover and applications," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2965–2982.

[45] arkworks contributors, "`arkworks` zksnark ecosystem," 2022. [Online]. Available: https://arkworks.rs

[46] W. Li, Z. Zhang, S. S. M. Chow, Y. Guo, B. Gao, X. Song, Y. Deng, and J. Liu, "Shred-to-shine metamorphosis in polynomial commitment evolution," Cryptology ePrint Archive, Paper 2025/1354, 2025. [Online]. Available: https://eprint.iacr.org/2025/1354

[47] Z. Zheng, X. Ji, T. Fang, F. Zhou, C. Liu, and G. Peng, "Batch-llm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching," *arXiv preprint arXiv:2412.03594*, 2024.

[48] V. Keršič, S. Karakatič, and M. Turkanović, "On-chain zero-knowledge machine learning: An overview and comparison," *Journal of King Saud University-Computer and Information Sciences*, vol. 36, no. 9, p. 102207, 2024.

[49] Z. Li, S. Vott, and B. Krishnamachari, "Ml2sc: Deploying machine learning models as smart contracts on the blockchain," in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2024, pp. 645–649.

[50] Q. Lin, Y. Zhu, X. Mei, L. Huang, J. Ma, K. He, Z. Peng, E. Cambria, and M. Feng, "Has multimodal learning delivered universal intelligence in healthcare? a comprehensive survey," *Information Fusion*, vol. 116, p. 102795, 2025.

[51] J. Ma, Q. Lin, Z. Jia, and M. Feng, "St-usleepnet: A spatial-temporal coupling prominence network for multi-channel sleep staging," *arXiv preprint arXiv:2408.11884*, 2024.

Appendix A. Definition of Functional Commitment

FC satisfies *perfect completeness* if for any function $f \in \mathcal{F}$ and any input $\vec{x}$, the following probability is 1:

$$\Pr\left[\begin{array}{r|r} \mathsf{VFR}(\mathsf{gp},\mathcal{C},\pi_c)=1 & \mathsf{gp}\leftarrow\mathsf{Setup}(1^\lambda,\mu) \\ \mathsf{Verify}(\mathsf{gp},\mathcal{C},\vec{x},f(\vec{x}),\pi)=1 & (\mathcal{C},\mathcal{D})\leftarrow\mathsf{Commit}(\mathsf{gp},f) \\ & \pi_f\leftarrow\mathsf{PFR}(\mathsf{gp},f,\mathcal{D}) \\ & (\vec{y},\pi)\leftarrow\mathsf{Eval}(\mathsf{pp},\mathcal{D},\vec{x},f) \end{array}\right]$$

It is *binding* if for any PPT adversary $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr\left[\begin{array}{r|r} b_0=b_1=1\wedge & \mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda,\mu) \\ f\neq f_1 & (\mathcal{C},f_0,\mathcal{D}_0,f_1,\mathcal{D}_1)\leftarrow\mathcal{A}(\mathsf{pp}) \\ & b_0\leftarrow\mathsf{Open}(\mathsf{gp},\mathcal{C},f_0,\mathcal{D}_0) \\ & b_1\leftarrow\mathsf{Open}(\mathsf{gp},\mathcal{C},f_1,\mathcal{D}_1) \end{array}\right]$$

The scheme satisfies *(knowledge) soundness* if PFR + Eval is an argument (of knowledge) for the relation defined as follows:

$$\left\{\mathbb{x}=\{\mathcal{C},\vec{x},\vec{y}\};\mathbb{w}=\{f,\mathcal{D}\}:\begin{array}{r} f\in\mathcal{F}\wedge f(\vec{x})=\vec{y} \\ \mathsf{Open}(\mathsf{gp},\mathcal{C},f,\mathcal{D})=1 \end{array}\right\}$$

PFR is used to prove the following relation:

$$\{\mathbb{x}=\{\mathcal{C}\};\mathbb{w}=\{f,\mathcal{D}\}:f\in\mathcal{F}\wedge\mathsf{Open}(\mathsf{gp},\mathcal{C},f,\mathcal{D})=1\}$$

A functional commitment scheme is *functional hiding* if for any input $\vec{x}$, any $f_0,f_1\in\mathcal{F}$ such that $f_0(\vec{x})=f_1(\vec{x})$, for any PPT adversary $\mathcal{A}$, the following probability is smaller than $\frac{1}{2}+\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{r|r} \mathcal{A}(\mathsf{gp},\mathcal{C},\pi_f,\pi)=b & \mathsf{gp}\leftarrow\mathsf{Setup}(1^\lambda,\mu) \\ & b\leftarrow\{0,1\} \\ & (\mathcal{C},\mathcal{D})\leftarrow\mathsf{Commit}(\mathsf{gp},f_b) \\ & \pi_f\leftarrow\mathsf{PFR}(\mathsf{gp},f,\mathcal{D}) \\ & (\vec{y},\pi)\leftarrow\mathsf{Eval}(\mathsf{pp},\mathcal{D},\vec{x},f_b) \end{array}\right]$$

Appendix B. Proof System for pR1CS

Protocol 3 presents the formal description of our proof system, generalized from Spartan [4].

Appendix C. Constraints of Instructions

C.1. Lookup

Assuming the $\vec{z}[i]$ is derived through Lookup, verifier needs to check

- all non-zero columns in $D[lr]$ are smaller than i, translated into

$$(\vec{adv}_1,\vec{t})\circ\vec{sel}_2\subseteq([n],\vec{max}_D)\cup(0,0)$$

Protocol 3. Amortized proof generation scheme for generalized R1CS

- Preprocess: $(\mathsf{gp}, \mathsf{vp}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{C}, T, \mathbb{w})$ takes input of security parameter λ, circuit $\mathcal{C}$, public table T and weights vector $\mathbb{p}$.
  1) Invoke $\mathsf{gp} \leftarrow \mathsf{PC.Setup}(1^\lambda)$.
  2) Parse circuit $\mathcal{C}$ as vectors $\vec{row}_M, \vec{col}_M, \vec{val}_M$, where $M \in \{A, B, C, D, E\}$, vectors $\vec{idx}_M$, where $M \in \{A, B, C\}$ and vector $\vec{tp}$.
  3) $\mathsf{com}_w \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{\mathbb{p}})$. $\mathsf{com}_{f,M} \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{f}_M)$ for all $f \in \{row, col, val\}, M \in \{A, B, C, D, E\}$. $\mathsf{com}_{idx,M} \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{idx}_M)$ for all $M \in \{A, B, C\}$. $\mathsf{com}_{tp} \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{tp})$. $\mathsf{com}_T \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{T})$
  4) Let $\mathsf{com}_\mathcal{C}$ be the concatenation of these polynomial commitments. Outputs $(\mathsf{pp}, \mathsf{com}_\mathcal{C})$.
- Prove: $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, \mathcal{C}, \mathbb{p}, \{\mathbb{x}_i\}_{i\in[N]})$
  1) Parse sparse matrices $A(\Gamma), B(\Gamma), C(\Gamma), D, E, \vec{tp}$ from $\mathcal{C}$. Compute $\{\mathbb{t}_i\}_{i\in[N]}$ from $\mathbb{p}$ and $\mathbb{x}$. For each $i \in [N]$, invoke $\mathsf{com}_{i,t} \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{\mathbb{t}}_i)$.
  2) Receive a shared random challenge $\gamma \in_R \mathbb{F}$ from verifiers. Compute $A_\gamma, B_\gamma, C_\gamma$ from $A(\Gamma), B(\Gamma), C(\Gamma)$ by substituting γ into them.
  3) Compute $\{\mathbb{a}_i\}_{i\in[N]}$ from $\mathbb{w}$, $\mathbb{x}$ and γ. For each $i \in [N]$, invoke $\mathsf{com}_{i,a} \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, \tilde{\mathbb{a}})$.
  4) For each $i \in [m]$, define $\vec{z}_i := 1||\mathbb{p}||\mathbb{x}_i||\mathbb{t}_i||\mathbb{a}_i$. Compute $\vec{m}_i := M\vec{z}_i$, where $M \in \{A_\gamma, B_\gamma, C_\gamma, D, E\}$.
  5) Receive a shared random challenge $\vec{r} \in \mathbb{F}^\mu$ from verifiers. Define $\vec{eq}_{\vec{r}}$ as the hypercube evaluations over $\vec{eq}_{\vec{r}}$.
  6) For each $i \in [N]$, run sumcheck to prove that $\sum_j \left(\vec{eq}_{\vec{r}} \circ (\vec{a}_i \circ \vec{b}_i - \vec{c}_i)\right)[j] = 0$, run LogUp to prove that $(\vec{d}_i, \vec{e}_i, \vec{tp}) \subseteq T$. Define the shared random challenges used for sumchecks are $\vec{r}_1$. These sumchecks will reduce the statements into the value of $\left\langle \vec{eq}_{\vec{r}_1}, \vec{m}_i \right\rangle = \vec{eq}_{\vec{r}_1}^T \cdot M \cdot \vec{z}_i = \left\langle \vec{eq}_{\vec{r}_1}^T \cdot M, \vec{z}_i \right\rangle \stackrel{?}{=} y_{m,i}$, where $m \in \{a, b, c, d, e\}$.
  7) Compute $\vec{m} := \vec{eq}_{\vec{r}_1}^T \cdot M$, where $M \in \{A, B, C, D, E\}$. For each $i \in [n]$, run sumcheck to prove $\langle \vec{m}, \vec{z}_i \rangle = y_{m,i}$. Define the shared random challenges used for sumchecks are $\vec{r}_2$. This will finally reduce the statement into $\tilde{m}(\vec{r}_2) \stackrel{?}{=} y_m$ and $\tilde{z}_i(\vec{r}_2) \stackrel{?}{=} y_{z,i}$ for some $y_m, y_{z,i} \in \mathbb{F}$.
  8) For each $i \in [n]$, invoke $\mathsf{PC.Eval}(\mathsf{pp}, \tilde{z}_i, \vec{r}_2)$ to prove the value of $\tilde{z}_i(\vec{r}_2) \stackrel{?}{=} y_{z,i}$.
  9) For $M \in \{A_\gamma, B_\gamma, C_\gamma, D, E\}$, prover proves $\tilde{m}(\vec{r}_2) = \tilde{M}(\vec{r}_1, \vec{r}_2) \stackrel{?}{=} y_m$ to all verifiers by invoking Protocol 1.

- $E[lr] = \{(i, 1)\}$, translated into

$$(\vec{adv}_1, \vec{1}) \circ \vec{sel}_2 \subseteq ([n], \vec{num}_E) \cup (0, 0)$$
$$(\vec{adv}_1, \vec{t}, \vec{1}) \circ \vec{sel}_2 \subseteq (\vec{row}_E, \vec{col}_E, \vec{val}_E) \cup (0, 0, 0)$$

C.2. Div

Constraining the Div operation is slightly more complicated, and we use 2 rows cr and $cr + 1$ to achieve this. Assuming $\vec{z}[i]$ is computed through Div, we set $C[cr]$ to show the correct computation of the division, and use $A[cr+1], B[cr+1], C[cr+1]$ to show that the remainder in $\vec{z}[aux] \in [d]$ is constrained correctly. The exact constraints and checks are as follows:

- All non-zero columns in $A[cr], B[cr]$ are smaller than i, tranalated into

$$(\vec{adv}_2, \vec{t}) \circ \vec{sel}_3 \subseteq ([n], \vec{max}_A) \cup (0, 0)$$
$$(\vec{adv}_2, \vec{t}) \circ \vec{sel}_3 \in ([n], \vec{max}_B) \cup (0, 0)$$

- $\langle A[cr], \vec{z} \rangle \cdot \langle B[cr], \vec{z} \rangle = d \cdot \vec{z}[i] + \vec{z}[aux]$, i.e. $C[cr] = \{(i, d), (aux, 1)\}$, translated into

$$(\vec{adv}_2, \vec{2}) \circ \vec{sel}_3 \subseteq ([n], \vec{num}_C) \cup (0, 0)$$
$$(\vec{adv}_2, \vec{t}, \vec{1}, \vec{0}) \circ \vec{sel}_3 \cup (\vec{adv}_2, \vec{adv}_4, \vec{adv}_1, \vec{0}) \circ \vec{sel}_3 \subseteq (\vec{row}_C, \vec{col}_C, \vec{val}_C, \vec{idx}_C) \cup (0, 0, 0, 0)$$

- $\vec{z}[aux + 1] = \vec{z}[aux] \cdot (d - 1 - z[aux])$, i.e. $A[cr + 1] = \{(aux, 1)\}$, $B[cr + 1] = \{(0, d - 1), (aux, -1)\}$, $C[cr + 1] = \{(aux + 1, 1)\}$, translated into

$$(\vec{adv}_2 + \vec{1}, \vec{1}) \circ \vec{sel}_3 \subseteq ([n], \vec{num}_A) \cup (0, 0)$$
$$(\vec{adv}_2 + \vec{1}, \vec{adv}_4, \vec{1}, \vec{0}) \circ \vec{sel}_3 \subseteq (\vec{row}_A, \vec{col}_A, \vec{val}_A, \vec{idx}_A) \cup (0, 0, 0, 0)$$
$$(\vec{adv}_2 + \vec{1}, \vec{2}) \circ \vec{sel}_3 \subseteq ([n], \vec{num}_B) \cup (0, 0)$$
$$(\vec{adv}_2 + \vec{1}, \vec{0}, \vec{adv}_4 - \vec{1}, \vec{0}) \circ \vec{sel}_3 \cup (\vec{adv}_2 + \vec{1}, \vec{adv}_4, -\vec{1}, \vec{0}) \circ \vec{sel}_3 \subseteq (\vec{row}_B, \vec{col}_B, \vec{val}_B, \vec{idx}_B) \cup (0, 0, 0, 0)$$
$$(\vec{adv}_2 + \vec{1}, \vec{1}) \circ \vec{sel}_3 \subseteq ([n], \vec{num}_C) \cup (0, 0)$$
$$(\vec{adv}_2 + \vec{1}, \vec{adv}_4 + \vec{1}, \vec{1}, \vec{0}) \circ \vec{sel}_3 \subseteq (\vec{row}_C, \vec{col}_C, \vec{val}_C, \vec{idx}_C) \cup (0, 0, 0, 0)$$

- $\vec{z}[aux + 1] \geq 0$, i.e. $D[lr] = \{(aux + 1, 1)\}$ and $\vec{tp}[lr] =$

GE0, translated into

$(\vec{adv}_3, \vec{1}) \circ \vec{sel}_3 \subseteq ([n], \vec{num}_D) \cup (0,0)$
$(\vec{adv}_3, \vec{adv}_4 + \vec{1}, \vec{1}, \vec{0}) \circ \vec{sel}_3 \subseteq (\vec{row}_D, \vec{col}_D, \vec{val}_D) \cup (0,0,0)$
$(\vec{adv}_3, \vec{\mathsf{GE0}}) \circ \vec{sel}_4 \subseteq ([n], \vec{tp})$

C.3. Matrix Multiplication

To prove $X \cdot Y = Z$, we write X, Y, Z as follows:

$$X = \left(\vec{x}^{(0)}, \cdots, \vec{x}^{(n-1)}\right)$$
$$Y^T = \left(\vec{y}^{(0)}, \cdots, \vec{y}^{(n-1)}\right)$$
$$Z = \begin{pmatrix} z_0 & \cdots & z_{k-1} \\ z_k & \cdots & z_{2k-1} \\ \vdots & \ddots & \vdots \\ z_{(m-1)\cdot k} & \cdots & z_{m\cdot k-1} \end{pmatrix}$$

According to Freivalds, to check $X \cdot Y = Z$, for random $\gamma \in \mathbb{F}$, let $\vec{\gamma}_1 = \left(\gamma^{0\cdot k}, \cdots, \gamma^{(m-1)\cdot k}\right)$, $\vec{\gamma}_2 = \left(\gamma^0, \cdots, \gamma^{k-1}\right)$, verifier checks

$$\left\langle X^T \cdot \vec{\gamma}_1, Y \cdot \vec{\gamma}_2 \right\rangle = \langle \vec{x}, \vec{y} \rangle \stackrel{?}{=} \sum_{i \in [m\cdot k]} z_i \cdot \gamma^i$$

where $\vec{x}, \vec{y} \in \mathbb{F}^n$ are defined for simplicity as

$$\vec{x}[i] := \left\langle \vec{\gamma}_1, \vec{x}^{(i)} \right\rangle \quad \vec{y}[i] := \left\langle \vec{\gamma}_2, \vec{y}^{(i)} \right\rangle \tag{8}$$

To prove that the matrix multiplication constraints are well formed, prover commits several auxiliary tables, such as MM table, InProd table and Constraint tables CstA, CstB, CstC. We first introduce columns of these tables and then introduce constraints.

Table columns. MM table records all matrix multiplications used in the whole program, where each row contains one multiplication. M, N, K represent matrix sizes, s_1, s_2, s_o represent starting positions of matrices. Specifically, the first matrix X's elements are in indices $[s_1, s_1 + mn)$, the second matrix Y's elements are in indices $[s_2, s_2 + nk)$, and the output matrix Z's elements are in indices $[s_o, s_o + mk)$; s_p represents starting position of vector $\vec{p} = (X^T \cdot \vec{\gamma}_1) \circ (Y \cdot \vec{\gamma}_2)$, i.e. $\vec{p} := \vec{z}[s_p..s_p + n - 1]$, $\vec{p}[i] = \vec{x}[i] \cdot \vec{y}[i]$. cr represents the constraint row of

$$\sum_{i \in [n]} \vec{p}[i] = \sum_{i \in [m\cdot k]} z_i \cdot \gamma^i \tag{9}$$

InProd table records inner products of matrix rows or columns with random vectors, which is used for checking Eq 8. Specifically, a matrix multiplication will generate precisely n rows in InProd, where each row constraints

$$\vec{p}[i] = \left\langle \vec{x}^{(i)}, \vec{\gamma}_1 \right\rangle \cdot \left\langle \vec{y}^{(i)}, \vec{\gamma}_2 \right\rangle \tag{10}$$

Table columns s_x represents starts of $x^{(i)}$, M represents length of $x^{(i)}$. Similarly, s_y and N represents starts and lengths of $y^{(i)}$. cnt represents the count of i, starting from 0 to $n-1$. res represents the position of $\vec{p}[i]$, i.e. $\vec{z}[res] = \vec{p}[i]$. cr represents the constraint row of Equation 10, namely

$$\langle \vec{z}, A_\gamma[cr] \rangle = \left\langle \vec{x}^{(i)}, \vec{\gamma}_1 \right\rangle$$
$$\langle \vec{z}, B_\gamma[cr] \rangle = \left\langle \vec{y}^{(i)}, \vec{\gamma}_2 \right\rangle$$
$$\langle \vec{z}, C_\gamma[cr] \rangle = \vec{z}[res]$$

Constraints tables CstA, CstB, CstC are used to further implement the inner product constraints. Specifically, each row in InProd will be extended to M rows in CstA, K rows in CstB, to compute $\left\langle \vec{x}^{(i)}, \vec{\gamma}_1 \right\rangle$ and $\left\langle \vec{y}^{(i)}, \vec{\gamma}_2 \right\rangle$ respectively. Each row in MM will be extended to n rows in CstA table and mk rows in CstC to check Equation 9. row and col represents the row and column index in the specific sparse matrix. cnt is used for counting the number of consecutive constraints, starting from 0. stp column is only included in CstA, specifying the step of index, as it will increase by k each term.

Table constraints. Next, we present all the constraints that these tables need to satisfy. Constraints table is the easiest. It only needs to check the following things:

- For each i, $\vec{cnt}[i]$ is either 0 or $\vec{cnt}[i] = \vec{cnt}[i-1] + 1$, $\vec{row}[i] = \vec{row}[i-1]$, $\vec{col}[i] = \vec{col}[i-1] + 1$, translated into

$$\vec{cnt} \circ (\mathsf{rsh}(\vec{cnt}) + \vec{1} - \vec{cnt}) = \vec{0}$$
$$\vec{cnt} \circ (\mathsf{rsh}(\vec{row}) - \vec{row}) = \vec{0}$$
$$\vec{cnt} \circ (\mathsf{rsh}(\vec{col}) + \vec{1} - \vec{col}) = \vec{0}$$

  For CstA, it additionally checks $\vec{cnt}[i] = 0$ or $\vec{stp}[i] = \vec{stp}[i-1]$, translated into

$$\vec{cnt} \circ (\mathsf{rsh}(\vec{stp}) - \vec{stp}) = \vec{0}$$

- All constraints are included in the sparse matrix:

$$\mathsf{CstA}.(\vec{row}, \vec{col}, vec(1), \vec{cnt} \circ \vec{stp}) \subseteq \left(\vec{row}_A, \vec{col}_A, \vec{val}_A, \vec{idx}_A\right)$$
$$\mathsf{CstB}.(\vec{row}, \vec{col}, vec(1), \vec{cnt}) \subseteq \left(\vec{row}_B, \vec{col}_B, \vec{val}_B, \vec{idx}_B\right)$$
$$\mathsf{CstC}.(\vec{row}, \vec{col}, vec(1), \vec{cnt}) \subseteq \left(\vec{row}_C, \vec{col}_C, \vec{val}_C, \vec{idx}_C\right)$$

InProd table is the bridge between the MM table and the Cst tables. It need to check the following:

- For each i, either $\vec{cnt}[i] = 0$ or $\vec{cnt}[i] = \vec{cnt}[i-1] + 1$, $\vec{M}[i] = \vec{M}[i-1]$, $\vec{K}[i] = \vec{K}[i-1]$, $\vec{s}_x[i] = \vec{s}_x[i-1] + \vec{M}[i]$, $\vec{s}_y[i] = \vec{s}_y[i-1] + \vec{K}[i]$, $\vec{res}[i] = \vec{res}[i-1] + 1$, translated into

$$\vec{cnt} \circ (\vec{cnt} - \mathsf{rsh}(\vec{cnt}) - \vec{1}) = \vec{0}$$
$$\vec{cnt} \circ (\vec{l}_x - \mathsf{rsh}(\vec{l}_x)) = \vec{0}$$
$$\vec{cnt} \circ (\vec{l}_y - \mathsf{rsh}(\vec{l}_y)) = \vec{0}$$
$$\vec{cnt} \circ (\vec{s}_x - \mathsf{rsh}(\vec{s}_x) + \vec{l}_x) = \vec{0}$$
$$\vec{cnt} \circ (\vec{s}_y - \mathsf{rsh}(\vec{s}_y) + \vec{l}_y) = \vec{0}$$
$$\vec{cnt} \circ (\vec{res} - \mathsf{rsh}(\vec{res}) - \vec{1}) = \vec{0}$$

- $(\vec{cr}, \vec{s}_x + \vec{M} - \vec{1}, \vec{M} - \vec{1}, \vec{l}_y) \subseteq \mathsf{CstA}$

- $(\vec{cr}, \vec{s}_y + \vec{K} - \vec{1}, \vec{K} - \vec{1}) \subseteq \mathsf{CstB}$
- $(\vec{cr}, r\vec{es}, \vec{1}, \vec{0}) \subseteq \left(r\vec{ow}_C, \vec{col}_C, \vec{val}_C, i\vec{dx}_C\right)$
- non-zero column number in $\vec{cr}[i]$ of A, B, C equals $\vec{M}[i]$, $\vec{K}[i]$ and 1 respectively:

$$(\vec{cr}, \vec{M}) \subseteq ([n], n\vec{um}_A)$$
$$(\vec{cr}, \vec{K}) \subseteq ([n], n\vec{um}_B)$$
$$(\vec{cr}, \vec{1}) \subseteq ([n], n\vec{um}_C)$$

MM table requires checking the following:

- $(\vec{s}_1 + \vec{M} \circ (\vec{N} - \vec{1}), \vec{M}, \vec{s}_2 + \vec{K} \circ (\vec{N} - \vec{1}), \vec{K}, \vec{N} - \vec{1}, \vec{s}_p + \vec{N} - \vec{1}) \subseteq \mathsf{InProd}.(\vec{s}_x, \vec{l}_x, \vec{s}_y, \vec{l}_y, \vec{cnt}, r\vec{es})$. This can ensure that all inner products are well-formed in InProd table.
- $(r\vec{ow}, \vec{prod} + \vec{N} - \vec{1}, \vec{N} - \vec{1}, \vec{0}) \subseteq \mathsf{CstA}$.
- $(r\vec{ow}, \vec{s}_o + \vec{M} \circ \vec{k} - \vec{1}, \vec{M} \circ \vec{k} - \vec{1}) \subseteq \mathsf{CstC}$.
- $(r\vec{ow}, \vec{0}, \vec{1}, \vec{0}) \subseteq (r\vec{ow}_B, \vec{col}_B, \vec{val}_B, i\vec{dx}_B)$.
- non-zero column number in $\vec{cr}[i]$ of A, B, C equals $\vec{N}[i]$, 1, and $\vec{M}[i] \cdot \vec{k}[i]$ respectively, by checking

$$(\vec{cr}, \vec{N}) \subseteq ([n], n\vec{um}_A)$$
$$(\vec{cr}, \vec{1}) \subseteq ([n], n\vec{um}_B)$$
$$(\vec{cr}, \vec{M} \circ \vec{k}) \subseteq ([n], n\vec{um}_C)$$

Appendix D. Proofs of Theorems

D.1. Theorem 1

Proof. For honest prover, the final output y should equal to

$$\sum_{k \in [n]} \vec{eq}_{\vec{z}_1}\left[r\vec{ow}[k]\right] \cdot \vec{eq}_{\vec{z}_2}\left[\vec{col}[k]\right] \cdot \vec{val}[k] \cdot \vec{\gamma}\left[i\vec{dx}[k]\right]$$

where n is the number of non-zero elements in the sparse matrix.

We first prove that if there exists $k \in [n]$ in the the committed vector $\vec{val}_r$ such that $\vec{val}_r[k] \neq \vec{eq}_{\vec{z}_1}[r\vec{ow}[k]]$, the probability it can pass the index lookup argument is only $\frac{O(n)}{|\mathbb{F}|}$, as

$$\left(r\vec{ow}[k], \vec{val}_r[k]\right) \notin \left([n], \vec{eq}_{\vec{z}_1}\right)$$

Similarly, we can ensure with $1 - \frac{O(n)}{|\mathbb{F}|}$ probability, for any $k \in [n]$, there has $\vec{val}_r[k] = \vec{eq}_{\vec{z}_1}[r\vec{ow}[k]]$, $\vec{val}_c = \vec{eq}_{\vec{z}_2}[\vec{col}[k]]$, $\vec{val}_i[k] = \vec{\gamma}[i\vec{dx}[k]]$. Given all these commitments are correct, using a sumcheck, their sum is correct with $1 - \frac{O(n)}{|\mathbb{F}|}$ probability. Using a union bound, the overall soundness error is $\frac{O(n)}{|\mathbb{F}|}$. □

D.2. Theorem 2

Proof. Assume the committed $\vec{z}_i$ doesn't satisfy $A_\gamma \vec{z}_i \circ B_\gamma \vec{z}_i = C_\gamma \vec{z}_i$, then

$$\sum_j \left(\vec{eq}_{\vec{r}} \circ (\vec{a}_i \circ \vec{b}_i - \vec{c}_i)\right)[j] \neq 0$$

Then after sumcheck in step 6, with $1 - \frac{O(n)}{|\mathbb{F}|}$ probability, there must be $\langle \vec{m}, \vec{z} \rangle \neq y_{m,i}$ for some $m \in \{a, b, c\}$. Assume $\langle \vec{m}, \vec{z} \rangle \neq y_{m,i}$ WLOG, then after sumcheck in step 7, there must be $\tilde{m}(\vec{r}_2) \neq y_m$ or $\tilde{z}_i(\vec{r}_2) \neq y_{z,i}$, which can't pass the PCS evaluation or sparse matrix evaluation scheme with $1 - \frac{O(n)}{|\mathbb{F}|}$ probability. Taking union bounds, the overall soundness error is $\frac{O(n)}{|\mathbb{F}|}$. □

D.3. Theorem 3

Proof. Here we present a sketch of the proof. We first demonstrate the correctness of $\vec{max}_M$ and $n\vec{um}_M$. The constraints can ensure all non-zero columns in the i-th row are smaller than $\vec{max}[i]$, and the total number of non-zero elements in the i-th row is not greater than $n\vec{um}[i]$. The reason that the number of non-zero elements can be smaller than $n\vec{um}[i]$ is because the prover may commit duplicate values.

Next, we discuss that the constraints of each instruction has been well-formed. It's quite straightforward that AddMult's constraints are well-formed if all conditions hold. Notice that all lookup input has vector $\vec{t}$ or $\vec{1}$, which doesn't include 0, so the union of all 0 tuple will not affect soundness. Similarly, constraints of Lookup and Div are also well-formed after checking the given conditions. The output-binding property of Lookup requires that for each lookup type tp, for any input $x \in \mathbb{F}$, there is at most one element y such that $(x, y, tp) \in T$. For a lookup type like GE0 where there is no explicit output, we can always set y to be 0.

MatMult is the hardest to analyze. The lookup from Table MM to InProd ensures that each row in MM will be translated into n consecutive rows in InProd, each of which records a correct inner product.

The lookup from InProd to constraint tables ensure that each row will be translated into l_x elements in matrix A, l_y elements in matrix B. Since it also checks $n\vec{um}_A$ and $n\vec{um}_B$, it ensures that the computation of the element-wise product of matrix A and matrix B is well-formed.

The lookup from MM to CstC ensures that the inner product of $\vec{\gamma}$ and the output matrix is well-formed. And the lookup from MM to CstA ensures that the sum of element-wise product is well-formed.

Overall, the matrix multiplication constraints are well formed. Considering that outputs of all instructions are computed from previous values, we can make sure the committed circuit corresponds to a program in $\mathcal{F}$. □

D.4. Theorem 4

Proof. Here, we present a proof sketch that for any function $f_1, f_2 \in \mathcal{F}$, such that $f_1(\vec{x}) = f_2(\vec{x})$, then they can't be distinguished.

Since the PCS is hiding, after FC.Commit, i.e. committing $A(\Gamma)$, $B(\Gamma)$, $C(\Gamma)$, D, E and vector $\vec{tp}$, no adversary can distinguish whether these commitments came from f_1 or f_2 because the PCS is hiding.

Since PFR scheme is a zkSNARK, no adversary can gain any information from PFR proof π_f. As the adversary can't distinguish f_1 and f_2 before seeing π_f, it still can't distinguish after seeing π_f.

Similarly, since Eval is a zkSNARK, no adversary can gain any information besides $f_1(\vec{x})$ after seeing Eval proof π. As $f_1(\vec{x}) = f_2(\vec{x})$, then no adversary can distinguish f_1 and f_2 after the whole process. □