Structured Matrix Constraint Systems for Architecture-Hiding Succinct Zero-Knowledge Proofs for Neural Networks

Mingshu Cong
Shenzhen University
Shenzhen, China

Sherman S. M. Chow
The Chinese University of Hong Kong
Sha Tin, N.T., Hong Kong

Tsz Hon Yuen
Monash University
Melbourne, Australia

Siu-Ming Yiu
The University of Hong Kong
Pok Fu Lam, Hong Kong

Abstract

Succinct zero-knowledge machine learning (zkML) uses zk succinct non-interactive arguments of knowledge (zkSNARKs) to prove neural-network (NN) computations with logarithmic-size proofs. However, general-purpose zkSNARKs do not scale in zkML because compiling matrix-heavy NNs into arithmetic circuits is memory-prohibitive. Existing zkML methods rely on rank-1 constraint systems (R1CS) to hide NN architectures while retaining succinctness. Removing circuit-based representations, it has remained unclear how to hide NN architectures without sacrificing succinctness.

Motivated by this gap, we introduce matrix-circuit satisfiability (Mat-Circ-SAT) and a high-dimensional variant of R1CS, termed high-dimensional R1CS (HD-R1CS), for Mat-Circ-SAT. Architecturally, HD-R1CS encodes NN architectures via sparse matrices whose dimensions scale with the number of matrices, rather than with the total number of scalar entries, as in R1CS. Notably, we present zkSMART (zero-knowledge sparse matrix argument via restructuring transform) as a zkSNARK protocol for HD-R1CS.

Compared to Evalyn (Asiacrypt '25), which hides the NN architecture using the proof-of-proof technique, zkSMART performs better in concrete prover time for deep NNs. More precisely, for NN computations with M matrices of size $n \times n$, we achieve $O(n^2M)$ prover time, $O(\log(nM))$ proof size and verifier time, and $O(n^2M)$ RAM usage with a small constant factor. Such asymptotic efficiency enables our protocol to scale to NNs with up to a billion parameters.

CCS Concepts

• **Security and privacy → Privacy-preserving protocols**.

Keywords

Zero-knowledge machine learning, zkSNARK, R1CS

1 Introduction

Zero-knowledge machine learning (zkML) [1, 5, 24, 27, 30, 33, 38, 42, 44, 46, 47, 57, 65–68] provides zero-knowledge proofs for neural-network (NN) computations. We focus on succinct zkML, where the proof size and verifier time are logarithmic (or constant) in the number of NN parameters. Succinctness matters when verification is resource-limited or widely replicated, *e.g.*, in blockchains, computation runs off-chain and a short proof is verified on-chain.

Email: minsheu.cong@gmail.com.

A central challenge for succinct zkML is to commit to NN architectures efficiently. Commitments to NN parameters with polynomial commitment schemes (PCSs) are now standard,[1] but architecture commitments in succinct zkML remain largely unexplored, primarily due to the lack of an efficient transformation from architectural information to polynomials that can be committed to by a PCS. We treat matrices as atomic computational units, so an NN with M matrix operations becomes a matrix circuit with M nodes, independent of the matrix dimension. Building on this abstraction, we formalize matrix-circuit satisfiability (Mat-Circ-SAT), extend R1CS to high-dimensional R1CS (HD-R1CS) for Mat-Circ-SAT, and present zkSMART as a zkSNARK protocol for HD-R1CS. Although zkML motivates this work, Mat-Circ-SAT also captures matrix programs beyond NNs, and we use zkML as the primary instantiation. Recent zkML work [65] also identifies NN architecture hiding as an "*exciting future research*" direction. This challenge matters:

① **Model privacy.** zkML threats arise not only at the cryptographic layer but also at the model layer. If the NN architecture (topology and operations) is public, adversaries can exploit structural priors to train surrogate models and leverage transferability for query-efficient black-box attacks. One example is adversarial examples, namely small norm-bounded perturbations that preserve semantics yet induce misclassification [36, 51, 58, 60]. Such attacks can undermine downstream decisions (*e.g.*, by increasing false positives or false negatives) and erode system trust. Accordingly, a zkML system should protect both NN parameters and NN architecture when the architecture is sensitive.

② **Succinctness.** To bind a proof to an NN, the verifier must incorporate sufficient information about the NN architecture. Without a succinct architecture commitment, the verifier must process the architectural description explicitly, so verifier time is at least linear in the number of operations. This limitation appears in existing non-architecture-hiding zkML systems. In particular, a broad class of divide-and-conquer zkML approaches, *e.g.*, [1, 5, 34, 42, 44, 46, 47, 57, 59, 65, 68], require the NN architecture to be public, since the verifier must know which operation is being verified at each step. These systems also generate proofs on an operation-by-operation basis, so proof size is linear in the number of operations. This is incompatible with resource-limited or widely replicated verification.

[1] Our concrete instantiation uses an elliptic-curve cryptography (ECC)-based PCS with a structured reference string (SRS). Transparent PCS alternatives (*e.g.*, FRI-like designs) avoid structured setup but target different proof-size and verifier-cost tradeoffs. Exploring these alternatives end-to-end is orthogonal to our main contribution.

Monolithic R1CS is impractical. An approach to hiding NN architectures is to compile the NN into an arithmetic circuit and apply general-purpose zkSNARKs, *e.g.*, [2, 8, 14, 18, 21–23, 29, 32, 35, 37, 49, 52–54, 69, 74]. Such systems encode arithmetic circuits into an intermediate representation (IR), such as rank-1 constraint systems (R1CS), which represent a circuit with three sparse matrices. Committing to these matrices can hide the circuit architecture. Early succinct zkML systems [27, 68] adopt this monolithic approach, which scales poorly and quickly becomes impractical.

Modern NNs often contain billions of parameters. Storing a billion parameters over a 256-bit field alone requires 32 GB of memory. Given that commodity servers typically provide 128–512 GB of RAM, zkML systems must keep memory overhead within a small constant factor of the parameter size. Unfortunately, directly converting matrix operations (*e.g.*, matrix multiplication or convolution) to scalar arithmetic yields superlinear circuit sizes, as observed in Zen [27] and pvCNN [68]. Although later systems mitigate this superlinearity, activation functions still introduce large constant factors in circuit size. As a result, monolithic R1CS-based approaches remain impractical at scale.

Sparse-matrix bottleneck in monolithic R1CS. In monolithic R1CS, committing to and opening the constraint matrices can dominate prover time and RAM. The bottleneck is not only the number of nonzero entries, but also the cost of representing sparsity and proving consistency of sparse accesses. In R1CS, the number of nonzero entries in these matrices scales with the circuit size. Under the monolithic circuit approach, the circuit size is at least proportional to the model parameter count. Hence, committing to and opening these matrices becomes prohibitive for large models. Existing sparse-arithmetization techniques, *e.g.*, Spartan [53] and Lasso [56], address sparsity via lookup proofs (see Appendix C.2). These methods require storing, committing to, and proving over several large auxiliary vectors, thereby incurring substantial RAM usage and prover-time overhead.

A further issue is that existing sparse-arithmetization targets *unstructured* sparsity and typically reduces sparse access to lookup-style consistency checks. In contrast, the large sparse matrices induced by HD-R1CS are *structured* (*e.g.*, Kronecker or block forms), which enables factorized MLE evaluation and avoids materializing large auxiliary lookup vectors. This structure enables an $O(M)$-scale cost for committing to and opening the architecture, avoiding a sparse-lookup bottleneck.

Prior attempts. Motivated by the need to hide NN architectures and by the impracticality of monolithic circuits, two recent works, pR1CS [38] and Evalyn [24], propose reduced-size circuit constructions. pR1CS applies Freivalds' algorithm [28] to matrix multiplications in NNs, allowing each $n \times n$ matrix multiplication to be verified as a length-n inner product. For an NN with M matrix operations on $n \times n$ matrices, the resulting circuit size is $O(Mn)$. Evalyn [24] adopts a proof-of-proof (PoP) strategy to compress proof size. It first generates a non-succinct operation-by-operation proof, and then applies a wrapping zkSNARK to attest that the non-succinct verifier accepts. Each $n \times n$ matrix operation corresponds to a verifier subcircuit of size $O(\log n)$, yielding a circuit size of $O(M \log n)$.

Evalyn's reliance on PoP introduces several limitations.
① Proving Fiat–Shamir hash computations incurs additional prover time linear in the NN depth, which becomes costly for deep NNs.
② Logarithmic proof size comes from an external compression layer, so the underlying zkML protocol is not intrinsically succinct.
③ The $O(M \log n)$ circuit size remains asymptotically suboptimal.

Beyond R1CS: matrix-level constraint systems. Existing zkSNARK frameworks operate at the scalar level, which inflates architecture encoding for matrix-native computation. The remainder of the paper specifies the HD-R1CS interface, gives PIOP reductions for its matrix relations, and proves that composing these reductions with a PCS yields a zkSNARK with the claimed properties.

Hiding NN architecture via compact HD-R1CS matrices. HD-R1CS encodes the NN architecture as sparse matrices that are substantially smaller than those in monolithic R1CS. Their nonzero patterns capture *matrix-level wiring* rather than scalar-level wiring. Consequently, these matrices have only $O(M)$ nonzero entries, and the prover time and memory to commit to and open the architecture scale linearly with M. This is asymptotically optimal, since any prover must at least read the $O(M)$-sized architecture description.

Structure-aware efficient prover. HD-R1CS matrices are large but structured (*e.g.*, Kronecker products or block matrices), so the prover can avoid materializing zero padding. This enables time- and memory-efficient execution.

Experiment. We implement zkSMART in Rust and generate proofs for a 1,024-layer NN with $1{,}024 \times 1{,}024$ parameter matrices. For this ∼1-billion-parameter model, peak RAM during proof generation is 104.93 GB, within the capacity of commercial servers. Prover time is 15.4 min, faster than Evalyn in our evaluation. Proof size is 367.50 KB, logarithmic in the parameter size and NN depth.

Scope and non-goals. This paper studies proof-system design for matrix programs under strict succinctness and architecture hiding. Our goal is to reduce the cryptographic overhead of proving structured matrix relations. We target settings where verification must be fast and succinct, and proving can run offline or in batch. This paper does not deliver an end-to-end verifiable-ML toolchain, nor does it claim latency-competitiveness against zkML systems that prioritize prover throughput or end-to-end latency via specialization or relaxed succinctness.[2] We evaluate a prototype on a large multi-layer perceptron (MLP) instance to quantify prover cost and to isolate the overhead of architecture commitments. Correctness is proved for the specified quantized computation. Accuracy under quantization is outside our scope. A reference encoder generates HD-R1CS instances for the evaluated model family. A full ONNX-to-HD-R1CS compiler is outside our scope.[3]

Our contributions. We make the following contributions:

[2] When zkSMART is the right tool: (i) the verifier is compute- or bandwidth-constrained and needs succinct proofs (*e.g.*, on-chain validation, audit logs, or thin clients), and (ii) revealing the model architecture is undesirable (*e.g.*, proprietary intellectual property, evasion resistance, or policy-controlled models). Systems such as zkLLM instead prioritize prover throughput and end-to-end latency.

[3] ONNX is a standard format that describes a neural network (NN) as a computational graph. A full ONNX-to-HD-R1CS frontend would parse the graph, infer tensor shapes, fix quantization parameters and semantics, and emit HD-R1CS matrices. Practical challenges include ONNX graph edge cases and enforcing a single quantization semantics that matches deployed inference bit-for-bit.

Table 1: Comparison of zkML protocols for NNs with M operations on $n \times n$ matrices

Protocol	Prover Time	Proof Size	Verifier Time	RAM Usage	Succinct[a]	NN Privacy[b]	Circuit Size
SafetyNets [33]	$O(M \log n)$	$O(Mn^2)$	$O(Mn^2)$	$O(Mn^2)$	○	○	-
Zen [27]	$O(Mn^3 \log n)^c$	$O(1)$	$O(1)$	$O(Mn^3)$	●	●	$O(Mn^3)$
pvCNN [68]	$O(Mn^4 \log n)^c$	$O(1)$	$O(1)$	$O(Mn^4)$	●	●	$O(Mn^4)$
vCNN [46]	$O(Mn^2 \log n)$	$O(M)$	$O(M)$	$O(Mn^2)$	◐	◐	-
KMC18 [44]	$O(Mn^2 \log n)$	$O(M \log n)$	$O(M \log n)$	$O(Mn^2)$	◐	◐	-
TrustlessDNN [42]	$O(Mn^2)$	$O(M \log n)$	$O(M \log n)$	$O(Mn^2)$	◐	◐	-
zkAudit [65]	$O(Mn^2)$	$O(M \log n)$	$O(M \log n)$	$O(Mn^2)$	◐	◐	-
zkCNN [47]	$O(Mn^2)$	$O(M \log^2 n)$	$O(Mn)^d$	$O(Mn^2)$	◐	◐	-
MSCProof [5]	$O(Mn^2)$	$O(M \log^2 n)$	$O(Mn)^d$	$O(Mn^2)$	◐	◐	-
zkLLM [57]	$O(Mn^2)$	$O(M \log n)$	$O(Mn)^d$	$O(Mn^2)$	◐	◐	-
pR1CS [38]	$O(Mn^2)$	$O(\log(Mn))$	$O(\log(Mn))$	$O(Mn^2)$	●	●	$O(Mn)$
Evalyn [24]	$O(Mn^2)$	$O(\log(Mn))$	$O(\log(Mn))$	$O(Mn^2)$	●	●	$O(M \log n)$
zkSMART	$O(Mn^2)$	$O(\log(Mn))$	$O(\log(Mn))$	$O(Mn^2)$	●	●	$O(M)$

[a] ●, ◐, and ○ represent constant or logarithmic (succinct), sublinear but not logarithmic, and linear proof size and verifier time, respectively.
[b] ●, ◐, and ○ represent architecture-and-parameter privacy, parameter privacy only, and no privacy, respectively.
[c] Complexities for unverified $n \times n$ matrix multiplication/convolution itself are $O(n^3)/O(n^4)$.
[d] zkCNN/zkLLM use Hyrax PCS, where verification time complexity is larger than the proof size.

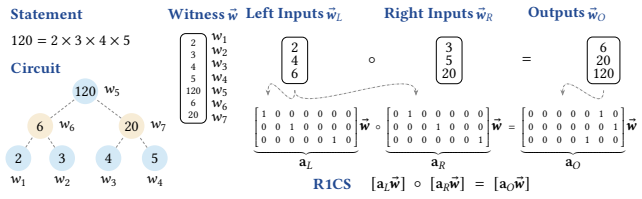


Figure 1: Example of R1CS relation encoding an arithmetic circuit's architecture as sparse matrices $\{a_L, a_R, a_O\}$

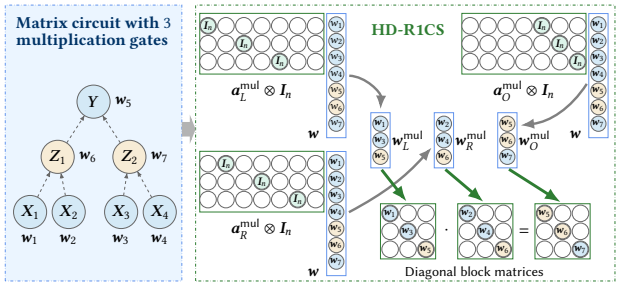


Figure 2: HD-R1CS for the statement $Y = (X_1X_2)(X_3X_4)$

- We introduce the first matrix-level constraint system that represents NN architectures as sparse matrices whose size depends only on the number of matrix operations.
- We show that NN architectures can be hidden with asymptotically optimal prover time and memory overhead $O(M)$.
- We instantiate architecture hiding by committing to the matrix wirings, and separate this overhead from other proving costs.
- We evaluate zkSMART's scaling on a large MLP instance to quantify prover time, memory, proof size, and verification time.

2 Technical Overview

This section summarizes the proof pipeline at a high level. We encode a matrix circuit as an HD-R1CS instance whose sparse topology matrices commit to the architecture. We then reduce verification to evaluations of multilinear extensions (MLEs) that can be proven efficiently by exploiting Kronecker and block-diagonal structure. Later sections instantiate these reductions in a zkSNARK via a standard PIOP+PCS composition.

2.1 From R1CS to HD-R1CS

R1CS for Circ-SAT. General-purpose zkSNARKs are commonly based on Circ-SAT. Circ-SAT asks for a witness $\vec{w}$ assigning values to circuit wires such that all multiplication constraints hold.

R1CS [12] is a popular intermediate representation for Circ-SAT. It encodes all multiplication constraints via a vector Hadamard (element-wise) product $\circ$:

$$[a_L\vec{w}] \circ [a_R\vec{w}] = [a_O\vec{w}]. \tag{1}$$

Here, $\vec{w}$ is the witness vector, and $\{a_L, a_R, a_O\}$ are sparse "rotating" matrices that align witness entries with the left inputs, right inputs, and outputs of multiplication gates. The matrices $\{a_L, a_R, a_O\}$ uniquely specify the circuit. For example, Figure 1 shows the circuit for $120 = 2 \times 3 \times 4 \times 5$. The prover sets intermediate wires to 6 and 20, and the verifier checks $2 \times 3 = 6$, $4 \times 5 = 20$, and $6 \times 20 = 120$. This yields $(2\ 4\ 6)^\intercal \circ (3\ 5\ 20)^\intercal = (6\ 20\ 120)^\intercal$. In a_L, a_R, and a_O, the nonzeros at $(3,6)$, $(3,7)$, and $(3,5)$ indicate that the 3rd multiplication gate reads $\vec{w}[6]$ and $\vec{w}[7]$ and outputs $\vec{w}[5]$.

Committing to the sparse matrices both hides the circuit architecture and enables $O(1)$-time verification from commitments, avoiding a linear scan of the circuit description.

HD-R1CS for Mat-Circ-SAT. We extend Circ-SAT to Mat-Circ-SAT for matrix computations by replacing scalar gates with matrix gates. For example, $Y = (X_1X_2)(X_3X_4)$ is witnessed by Z_1, Z_2 such that $X_1X_2 = Z_1$, $X_3X_4 = Z_2$, and $Z_1Z_2 = Y$. In R1CS, $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$ align witness entries with the left inputs, right inputs, and outputs of scalar multiplication gates. Each row contains a single nonzero entry, selecting one witness entry per gate. HD-R1CS preserves this role but lifts it to $n\times n$ blocks by replacing each scalar entry in Eq. (1) with an $n\times n$ matrix block. Accordingly, $\{0, 1\}$ entries are interpreted as the zero block $\mathbf{0} \in \mathbb{F}^{n\times n}$ and the identity block $\boldsymbol{I}_n \in \mathbb{F}^{n\times n}$. For the running example, the sparse patterns of $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$ match those in the scalar R1CS instance for $120 = 2 \times 3 \times 4 \times 5$.

Kronecker lifting. In R1CS, multiple scalar multiplication constraints are collectively verified by a single vector Hadamard product. Analogously, in HD-R1CS, multiple matrix multiplications can be aggregated into a single block-structured matrix equation. For the three constraints $X_1X_2 = Z_1 \ \wedge \ X_3X_4 = Z_2 \ \wedge \ Z_1Z_2 = Y$, we form the diagonal-block matrix multiplication:

$$\mathsf{Diag}_n\left[X_1^\top, X_3^\top, \mathbf{0}\right]^\top \cdot \mathsf{Diag}_n\left[X_2^\top, X_4^\top, Z_2^\top\right]^\top = \mathsf{Diag}_n\left[Z_1^\top, Z_2^\top, Y^\top\right]^\top. \tag{2}$$

Here, $\mathsf{Diag}_n([\boldsymbol{A}_1^\top, \ldots, \boldsymbol{A}_m^\top]^\top)$ denotes the $mn \times mn$ block-diagonal matrix with diagonal blocks $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ (Figure 2). As in R1CS, the witness packs all gate values. In HD-R1CS, the witness is a block-matrix witness $\boldsymbol{w} \in \mathbb{F}^{(7n)\times n}$ that stacks all $n \times n$ gate matrices:

$$\boldsymbol{w} = (X_1^\top \ X_2^\top \ X_3^\top \ X_4^\top \ Z_1^\top \ Z_2^\top \ Y^\top)^\top. \tag{3}$$

In R1CS, multiplying the witness $\vec{\boldsymbol{w}}$ by $\boldsymbol{a}_{L/R/O}$ rotates scalar entries. In HD-R1CS, $\boldsymbol{w}$ is a $(7n) \times n$ matrix while $\boldsymbol{a}_{L/R/O}$ are 3×7 matrices. To rotate and align $n\times n$ blocks, we lift the sparse matrices via the Kronecker product $\boldsymbol{a}_{L/R/O} \otimes \boldsymbol{I}_n$. For example:

$$\boldsymbol{a}_O \otimes \boldsymbol{I}_n = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}_n \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}_n & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{4}$$

Multiplying $\boldsymbol{w}$ by $(\boldsymbol{a}_{L/R/O} \otimes \boldsymbol{I}_n)$ rotates and aligns its blocks with the left inputs, right inputs, and outputs of matrix multiplication gates. We never materialize $\boldsymbol{a}_{L/R/O} \otimes \boldsymbol{I}_n$ or the resulting diagonal-block matrices, and instead prove the required MLE evaluations by factoring their structure. Combining the above, the following HD-R1CS relation collectively verifies all matrix multiplications:

$$\mathsf{Diag}_n[(\boldsymbol{a}_L \otimes \boldsymbol{I}_n)\boldsymbol{w}] \cdot \mathsf{Diag}_n[(\boldsymbol{a}_R \otimes \boldsymbol{I}_n)\boldsymbol{w}] = \mathsf{Diag}_n[(\boldsymbol{a}_O \otimes \boldsymbol{I}_n)\boldsymbol{w}].$$

HD-R1CS also supports other matrix operations, and we defer these extensions to later sections.

Hiding matrix circuit architecture. In HD-R1CS, the matrices $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$ encode the matrix-circuit architecture. Committing to these matrices commits to and hides the circuit architecture. For succinct verification, the verifier takes as input only the commitments to $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$, rather than the matrices themselves.

2.2 Structured MLE Reduction in HD-R1CS

At first glance, Kronecker lifting and diagonal-block transformations appear to reintroduce large sparse matrices. For example, the Kronecker-lifted matrix $\boldsymbol{a}_O \otimes \boldsymbol{I}_n$ in Eq. (4) contains $O(3n)$ nonzero blocks. However, the large matrices that arise in HD-R1CS are highly structured. Their multilinear extensions (MLEs) can therefore be evaluated efficiently by factoring them, without resorting to lookup proofs that target unstructured sparse matrices in R1CS.

Unstructured sparse matrices in R1CS. When NNs are transformed into R1CS, the resulting large sparse matrices typically lack exploitable algebraic structure. Spartan [53] proposes a sparse-matrix arithmetization that verifies MLE values via lookup proofs. In a toy example with $\boldsymbol{S} = \begin{pmatrix} 0 & 0 \\ 1 & 2 \end{pmatrix}$, one can write

$$\widehat{\boldsymbol{S}}(x_1, x_2) = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \circ \begin{pmatrix} x_1 \\ x_1 \end{pmatrix} \circ \begin{pmatrix} 1 - x_2 \\ x_2 \end{pmatrix}.$$

Lookup proofs enforce consistent term selection in this Hadamard decomposition of $\widehat{\boldsymbol{S}}$ into short vectors, but their overhead becomes prohibitive once the number of nonzero entries reaches billions.

Structured sparse matrices in HD-R1CS. The large sparse matrices in HD-R1CS (*e.g.*, $\boldsymbol{a}_O \otimes \boldsymbol{I}_n$ in Eq. (4)) are structured and admit factorized MLE evaluation. For example, $\widehat{\boldsymbol{S} \otimes \boldsymbol{I}_2}$ factors as:

$$\begin{aligned}\widehat{\boldsymbol{S}\otimes\boldsymbol{I}_2}(x_1, x_2, x_3, x_4) =& [x_1(1 - x_2) + 2x_1x_2] \cdot [(1 - x_3)(1 - x_4) + x_3x_4] \\ =& \widehat{\boldsymbol{S}}(x_1, x_2) \cdot \widehat{\boldsymbol{I}_2}(x_3, x_4).\end{aligned}$$

Verifying the large evaluation $\widehat{\boldsymbol{S} \otimes \boldsymbol{I}_2}(x_1, x_2, x_3, x_4)$ is reduced to verifying evaluations of the smaller factors $\widehat{\boldsymbol{S}}(x_1, x_2)$ and $\widehat{\boldsymbol{I}_2}(x_3, x_4)$. Similar structure-aware arguments apply to diagonal-block matrices. The large-scale matrices induced by HD-R1CS admit such factorizations and can therefore be processed efficiently.

2.3 Structured Floating-Point Quantization

Due to finite-precision constraints, floating-point computations are typically quantized into integer operations in cryptographic protocols. Quantized neural networks (QNNs) [41, 71] provide a standard framework for this transformation. Under quantization, each floating-point matrix $\boldsymbol{a}^{\mathsf{f}} \in \mathbb{R}^{n\times n}$ is represented as $\boldsymbol{a}^{\mathsf{f}} = \epsilon_a^{\mathsf{f}} \cdot \boldsymbol{a}$, where $\boldsymbol{a} \in \mathbb{F}^{n\times n}$ is a k-bit (*e.g.*, 8-bit) integer matrix and $\epsilon_a^{\mathsf{f}} \in \mathbb{R}$ is a scaling factor (*e.g.*, $\epsilon_a^{\mathsf{f}} = 1/127$ for 8-bit quantization). With this representation, floating-point matrix multiplication $\boldsymbol{c}^{\mathsf{f}} = \boldsymbol{a}^{\mathsf{f}}\boldsymbol{b}^{\mathsf{f}}$ is implemented over integer arithmetic as $\boldsymbol{c} = \lfloor \theta_{a,b,c} \cdot (\boldsymbol{ab}) \rfloor$, where $\theta_{a,b,c} = \epsilon_a^{\mathsf{f}}\epsilon_b^{\mathsf{f}}/\epsilon_c^{\mathsf{f}}$. Here, $\lfloor\cdot\rfloor$ denotes the deterministic rounding specified by the quantization parameters, and its enforcement is captured by explicit constraints detailed later. This reduces floating-point computation to integer computations followed by a rescaling.

Quantized HD-R1CS. The HD-R1CS formulation extends to quantized matrix computations. Consider the floating-point relation $Y^{\mathsf{f}} = X_1^{\mathsf{f}}X_2^{\mathsf{f}}X_3^{\mathsf{f}}X_4^{\mathsf{f}}$. Let ϵ_i^{f} denote the scaling factors for $i \in \{X_1, X_2, X_3, X_4, Z_1, Z_2, Y\}$. Define $\theta_{i,j,k} = \epsilon_{X_i}^{\mathsf{f}}\epsilon_{X_j}^{\mathsf{f}}/\epsilon_{Z_k}^{\mathsf{f}}$ and $\theta_{i,j,Y} = \epsilon_{Z_i}^{\mathsf{f}}\epsilon_{Z_j}^{\mathsf{f}}/\epsilon_Y^{\mathsf{f}}$. Then the quantized counterpart of Eq. (2) is:

$$\mathsf{Diag}_n\begin{pmatrix} Z_1 \\ Z_2 \\ Y \end{pmatrix} = \left\lfloor \begin{bmatrix} \theta_{1,2,1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \theta_{3,4,2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \theta_{1,2,Y} \end{bmatrix} \otimes \boldsymbol{I}_n \right\rfloor \cdot \mathsf{Diag}_n\begin{pmatrix} X_1 \\ X_3 \\ Z_1 \end{pmatrix} \cdot \mathsf{Diag}_n\begin{pmatrix} X_2 \\ X_4 \\ Z_2 \end{pmatrix} \rfloor.$$

This relation preserves the Kronecker structure of Eq. (2). Thus, quantized floating-point matrix circuits can be handled within HD-R1CS under essentially the same proof framework.

3 Preliminaries

We consider matrix programs with M operations on input and output matrices. All matrices have dimensions at most $n \times n$ and entries that fit in k bits. We embed these k-bit integers into a finite field $\mathbb{F}$. We write $\mathbb{F}^* \coloneqq \mathbb{F} \setminus \{0\}$. Matrices and vectors are denoted by bold symbols, with $\vec{\cdot}$ indicating column vectors and $\vec{\cdot}^{\mathsf{T}}$ indicating row vectors, *e.g.*, $\boldsymbol{a}$, $\vec{\boldsymbol{l}}$, and $\vec{\boldsymbol{r}}^{\mathsf{T}}$. For a matrix $\boldsymbol{a} \in \mathbb{F}^{n\times n}$, we write $\boldsymbol{a} = \{a_{ij}\}$ for its entries. We use $[n]$ to denote $\{1, \ldots, n\}$. We use $*$ to denote a matrix operation. For $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$, $*$ may represent addition $+$, multiplication $\cdot$, Hadamard product $\circ$, or convolution $\boldsymbol{c} = \mathsf{Conv}(\boldsymbol{a}, \boldsymbol{b})$. For $\boldsymbol{c} = *\,\boldsymbol{a}$, $*$ may represent transpose $\boldsymbol{c} = \boldsymbol{a}^{\mathsf{T}}$ or an element-wise activation $\boldsymbol{c} = \phi(\boldsymbol{a}) \coloneqq \{\phi(a_{ij})\}$ for $\phi\colon \mathbb{R} \to \mathbb{R}$.

A block matrix is a matrix partitioned into submatrices called blocks. Block-matrix multiplication treats blocks as scalar elements. For instance:

$$\begin{pmatrix} \boldsymbol{a}_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{21} & \boldsymbol{a}_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{b}_{11} & \boldsymbol{b}_{12} \\ \boldsymbol{b}_{21} & \boldsymbol{b}_{22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{a}_{11}\boldsymbol{b}_{11} + \boldsymbol{a}_{12}\boldsymbol{b}_{21} & \boldsymbol{a}_{11}\boldsymbol{b}_{12} + \boldsymbol{a}_{12}\boldsymbol{b}_{22} \\ \boldsymbol{a}_{21}\boldsymbol{b}_{11} + \boldsymbol{a}_{22}\boldsymbol{b}_{21} & \boldsymbol{a}_{21}\boldsymbol{b}_{12} + \boldsymbol{a}_{22}\boldsymbol{b}_{22} \end{pmatrix},$$

where $\boldsymbol{a}_{ij}$ and $\boldsymbol{b}_{ij}$ are blocks.

The Kronecker product $\otimes$ of matrices $\boldsymbol{a} \in \mathbb{F}^{m\times n}$ and $\boldsymbol{b}$ is the block matrix:

$$\boldsymbol{a} \otimes \boldsymbol{b} \coloneqq \begin{pmatrix} a_{11}\boldsymbol{b} & \cdots & a_{1n}\boldsymbol{b} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{b} & \cdots & a_{mn}\boldsymbol{b} \end{pmatrix}.$$

The mixed-product property states that for compatible dimensions:

$$(\boldsymbol{a} \otimes \boldsymbol{b})(\boldsymbol{c} \otimes \boldsymbol{d}) = (\boldsymbol{a}\boldsymbol{c}) \otimes (\boldsymbol{b}\boldsymbol{d}).$$

We convert a dense block vector $\boldsymbol{w}$ with M blocks $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_M$ into a diagonal-block matrix by:

$$\mathsf{Diag}_n(\boldsymbol{w}) \coloneqq \begin{pmatrix} \boldsymbol{w}_1 & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{w}_M \end{pmatrix}.$$

MLE and PCS. We identify an $m \times n$ matrix with its multilinear extension (MLE) over $\{0,1\}^{\log_2 m + \log_2 n}$. We assume m and n are powers of two, or otherwise pad the matrix with zeros to the next powers of two. The MLE of a matrix $\boldsymbol{a}$ is denoted by $\widehat{\boldsymbol{a}}(\boldsymbol{x})$.

Definition 1 (MLE [59]). *The MLE $\widehat{\boldsymbol{a}}(\boldsymbol{x})$ of a matrix $\boldsymbol{a} \in \mathbb{F}^{m\times n}$ is the unique multilinear polynomial $\mathbb{F}^{\log_2 m + \log_2 n} \mapsto \mathbb{F}$ such that:*

$$\widehat{\boldsymbol{a}}(\boldsymbol{i}, \boldsymbol{j}) = a_{ij}, \quad \forall \boldsymbol{i} \in \{0,1\}^{\log_2 m}, \boldsymbol{j} \in \{0,1\}^{\log_2 n},$$

where $\boldsymbol{i}$ and $\boldsymbol{j}$ are the bit decompositions of $i \in [m]$ and $j \in [n]$, respectively.

Existing research [13, 19, 22, 45, 53] shows that MLE evaluation corresponds to a vector-matrix-vector product.

Lemma 1. *The MLE $\widehat{\boldsymbol{a}}(\cdot)$ of a matrix $\boldsymbol{a} \in \mathbb{F}^{m\times n}$ satisfies:*

$$\widehat{\boldsymbol{a}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \mathsf{vec}(\boldsymbol{a}) \bullet (\vec{\boldsymbol{l}} \otimes \vec{\boldsymbol{r}}) = \vec{\boldsymbol{l}}^{\mathsf{T}} \boldsymbol{a} \vec{\boldsymbol{r}}, \tag{5}$$

where $\vec{\boldsymbol{l}} \in \mathbb{F}^m$ and $\vec{\boldsymbol{r}} \in \mathbb{F}^n$ are computed from $\boldsymbol{x}_l \in \mathbb{F}^{\log_2 m}$ and $\boldsymbol{x}_r \in \mathbb{F}^{\log_2 n}$ as:

$$\begin{aligned} \vec{\boldsymbol{l}} &= (1 - x_{l1}, x_{l1}) \otimes \cdots \otimes (1 - x_{l\log_2 m}, x_{l\log_2 m}), \\ \vec{\boldsymbol{r}} &= (1 - x_{r1}, x_{r1}) \otimes \cdots \otimes (1 - x_{r\log_2 n}, x_{r\log_2 n}). \end{aligned} \tag{6}$$

This lemma will be used repeatedly throughout this paper.

Due to the correspondence between matrices and multilinear polynomials, a polynomial commitment scheme can also serve as a matrix commitment scheme. We define the PCS in an equivalent matrix notation.

Definition 2 (Multilinear polynomial commitment scheme). *A multilinear PCS for matrix commitments is a tuple of algorithms:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, m, n)$ *takes as input (m, n) and produces public parameters* pp*;*
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{a})$ *(or $\mathsf{Com}(\boldsymbol{a})$ for short) takes as input $\boldsymbol{a} \in \mathbb{F}^{m\times n}$ and produces a commitment C;*
- $\mathsf{bit} \leftarrow \mathsf{Open}(\mathsf{pp}, C, (m, n), v, \boldsymbol{x}\colon \boldsymbol{a})$ *is a protocol between a probabilistic polynomial-time prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{P}$ and $\mathcal{V}$ hold C, (m, n), $v \in \mathbb{F}$, and $\boldsymbol{x} \in \mathbb{F}^{\log_2 m + \log_2 n}$. $\mathcal{P}$ additionally knows $\boldsymbol{a} \in \mathbb{F}^{m\times n}$. $\mathcal{P}$ aims to convince $\mathcal{V}$ that $\widehat{\boldsymbol{a}}(\boldsymbol{x}) = v$. At the end of* Open*, $\mathcal{V}$ outputs* $\mathsf{bit} \in \{\mathsf{true}, \mathsf{false}\}$.

PIOP and sumcheck. Polynomial interactive oracle proofs (PIOPs) [9, 18] reduce a relation $\mathcal{R}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_s)$, via interactive queries, to checking individual evaluations $\widehat{\boldsymbol{a}}_i(\boldsymbol{x}_i)$ at random challenge points $\boldsymbol{x}_i$ for $i \in [s]$. Modern zkSNARK systems commonly combine a PIOP with a multilinear PCS.

The sumcheck protocol [48], a widely used PIOP primitive [21, 26, 34, 35, 53, 59, 62–64, 69, 75], reduces verification of $\sum_{\boldsymbol{x}\in\{0,1\}^\mu} \psi(\boldsymbol{x})$ to checking a claimed value $v = \psi(\boldsymbol{x})$ at a random challenge point $\boldsymbol{x}$. Here, ψ is a low-degree polynomial with μ variables [59]. The evaluation $\psi(\boldsymbol{x})$ is then proven via the Open algorithm of a PCS. Sumcheck protocols on MLEs can verify inner products, Hadamard products, and matrix multiplication, as detailed in Appendix B.2. We use these atomic protocols in zkSMART.

zkSNARK. See Appendix B.1 for standard definitions for arguments of knowledge, zkSNARK, perfect completeness, knowledge soundness, zero-knowledge, and Fiat–Shamir transformation.

Neural Networks. Neural networks are typically organized in layers, where data flows from an input layer through intermediate layers to an output layer. A standard layer with activation ϕ is represented by:

$$Y = \phi(\boldsymbol{W}X + \boldsymbol{\beta}),$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the layer input and output and $\boldsymbol{W}$ and $\boldsymbol{\beta}$ are parameters.

4 HD-R1CS:Constraint System for Mat-Circ-SAT

This section defines Mat-Circ-SAT and HD-R1CS, our matrix-level analogs of Circ-SAT and R1CS. We encode matrix-circuit wiring by three row-sparse $\{0,1\}$ matrices $\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O$. We encode the gate partition by operation type via selector matrices $\{\boldsymbol{b}^{(t)}\}_{t=0}^{T}$. We route $n \times n$ blocks by Kronecker lifting with $\boldsymbol{I}_n$, and we batch constraints into $T + 1$ equations.

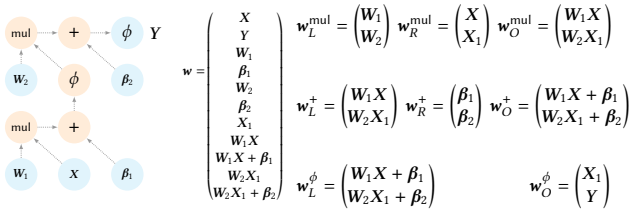


Figure 3: Example matrix circuit for a two-layer NN given by: $Y = \phi(W_2 \cdot \phi(W_1 \cdot X + \beta_1) + \beta_2)$, showing the witness matrix w and $\{w_L^{(t)}, w_R^{(t)}, w_O^{(t)}\}_{t\in\{\text{mul},+,\phi\}}$.

4.1 Matrix Circuit Satisfiability

General-purpose zkSNARKs support deterministic computations by compiling them into arithmetic circuits. The induced Circ-SAT problem requires assigning values to each gate so that all gate constraints are satisfied. Circ-SAT is then compiled into intermediate representations, including R1CS [12], quadratic arithmetic programs (QAP) [32], Plonk(ish) arithmetization [21, 29], algebraic intermediate representation [7, 10], and customizable constraint systems [55]. All of these IRs are scalar-based, and they are therefore ill-suited for matrix-intensive computations.

We depart from scalar IRs by lifting arithmetic circuits to matrix circuits. Apart from zkML, Mat-Circ-SAT also captures matrix programs beyond NNs; we use zkML as our primary instantiation. In zkML, NN inference and gradient-based training are compositions of matrix operations. Figure 3 shows a two-layer NN compiled into a matrix circuit, with nodes labeled by operations in $\mathbb{T} = \{\text{mul}, +, \phi\}$.

Definition 3 (Matrix Circuit). *Given a predefined set of matrix operations $\mathbb{T}$, a* matrix circuit *is a directed acyclic graph whose nodes represent input/output (I/O) matrices or operations drawn from $\mathbb{T}$.*

Definition 4 (Mat-Circ-SAT). *The matrix circuit satisfiability problem is to find an assignment of matrices to all intermediate nodes in a given matrix circuit such that the constraint induced by each operation node is satisfied on its assigned inputs and outputs.*

4.2 Sparse-Matrix Architecture Encoding

Encoding the topology. Consider a matrix circuit with M gates, consisting of $M^{(0)}$ I/O gates and $M - M^{(0)}$ operation gates whose types lie in $\mathbb{T}$. Let $|\mathbb{T}| = T$, and let $M^{(t)}$ denote the number of gates of operation type $t \in [T]$. Then $M^{(0)} + \sum_{t=1}^{T} M^{(t)} = M$.

We order the M gates such that I/O gates appear first, followed by type 1, type 2, and so on. Let the matrix assignment at gate i be $\boldsymbol{w}_i$, yielding the witness sequence:

$$\underbrace{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{M^{(0)}}}_{\text{I/O}}, \underbrace{\boldsymbol{w}_{M^{(0)}+1}, \ldots, \boldsymbol{w}_{M^{(0)}+M^{(1)}}}_{\text{type 1}}, \ldots, \underbrace{\boldsymbol{w}_{M-M^{(T)}+1}, \ldots, \boldsymbol{w}_M}_{\text{type } T}. \quad (7)$$

We encode the circuit wiring with sparse matrices $\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O \in \{0,1\}^{M\times M}$. The first $M^{(0)}$ rows are zero. For each operation gate $i > M^{(0)}$, the i-th row of $\boldsymbol{a}_L$ has a single 1 that points to the left-input gate index. For each binary operation, the i-th row of $\boldsymbol{a}_R$ has a single 1, and for each unary operation it is all zeros. For each operation gate $i > M^{(0)}$, the i-th row of $\boldsymbol{a}_O$ has a single 1 that points to the output gate index. As in R1CS, the triple $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$ uniquely specifies the circuit wiring, as illustrated in Figure 2.

Encoding the operation partition. We also hide the operation type assigned to each gate. At the same time, our relation is written per operation type, so we need a way to select the type-t rows without opening the partition. To this end, we define $T + 1$ sparse selector matrices $\{\boldsymbol{b}^{(t)} \in \{0,1\}^{M\times M}\}_{t=0}^{T}$. For $t \in \{0\} \cup [T]$, $\boldsymbol{b}^{(t)}$ extracts the $M^{(t)}$ rows associated with type t from an M-row matrix, and it pads the remaining rows with zeros. Concretely, $\boldsymbol{b}^{(0)}$ selects the first $M^{(0)}$ rows, and for each $t \in [T]$, $\boldsymbol{b}^{(t)}$ selects the contiguous block of $M^{(t)}$ rows corresponding to type t under the ordering in Eq. (7). Equivalently, each $\boldsymbol{b}^{(t)}$ is an $M \times M$ matrix with a single identity block of size $M^{(t)} \times M^{(t)}$ in the columns of type t, and zeros elsewhere. The family $\{\boldsymbol{b}^{(t)}\}_{t=0}^{T}$ uniquely encodes the partition $\{M^{(t)}\}_{t=0}^{T}$. Consequently, committing to these selector matrices implicitly commits to the operation type of every gate.

4.3 Blockwise Routing with Sparse Matrices

Block alignment. We stack $\{\boldsymbol{w}_i\}_{i=1}^{M}$ into a witness matrix:

$$\boldsymbol{w} = \left(\boldsymbol{w}_1^{\mathsf{T}}, \ldots, \boldsymbol{w}_M^{\mathsf{T}}\right)^{\mathsf{T}} \in \mathbb{F}^{(Mn)\times n}.$$

For exposition, we present the homogeneous $n \times n$ case in which each $\boldsymbol{w}_i$ is an $n \times n$ block.

The wiring matrices $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$ specify, for each gate, the indices of its left input, right input, and output. To apply them to the block matrix $\boldsymbol{w}$, we lift each by taking the Kronecker product with $\boldsymbol{I}_n \in \{0,1\}^{n\times n}$, obtaining:

$$\boldsymbol{w}_L = (\boldsymbol{a}_L \otimes \boldsymbol{I}_n)\boldsymbol{w}, \quad \boldsymbol{w}_R = (\boldsymbol{a}_R \otimes \boldsymbol{I}_n)\boldsymbol{w}, \quad \boldsymbol{w}_O = (\boldsymbol{a}_O \otimes \boldsymbol{I}_n)\boldsymbol{w}. \quad (8)$$

Each resulting $(Mn) \times n$ block matrix aligns the left input, right input, or output for all M gates.

Operation-wise selection. Matrices $\boldsymbol{w}_L, \boldsymbol{w}_R, \boldsymbol{w}_O$ aggregate gates of all operation types. To select the inputs and outputs for operation type t, we multiply by the Kronecker-lifted selector $(\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)$:

$$\boldsymbol{w}_L^{(t)} = (\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)\boldsymbol{w}_L, \boldsymbol{w}_R^{(t)} = (\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)\boldsymbol{w}_R, \boldsymbol{w}_O^{(t)} = (\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)\boldsymbol{w}_O. \quad (9)$$

Each $\boldsymbol{w}_{L/R/O}^{(t)}$ contains exactly the blocks for type t, and all other blocks are zero.

Applying the mixed-product property to Eqs. (8) and (9) yields, for each $t \in \{0\} \cup [T]$:

$$\begin{aligned} \boldsymbol{w}_L^{(t)} &= (\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)[(\boldsymbol{a}_L \otimes \boldsymbol{I}_n)\boldsymbol{w}] = [(\boldsymbol{b}^{(t)}\boldsymbol{a}_L) \otimes \boldsymbol{I}_n]\boldsymbol{w}, \\ \boldsymbol{w}_R^{(t)} &= (\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)[(\boldsymbol{a}_R \otimes \boldsymbol{I}_n)\boldsymbol{w}] = [(\boldsymbol{b}^{(t)}\boldsymbol{a}_R) \otimes \boldsymbol{I}_n]\boldsymbol{w}, \\ \boldsymbol{w}_O^{(t)} &= (\boldsymbol{b}^{(t)} \otimes \boldsymbol{I}_n)[(\boldsymbol{a}_O \otimes \boldsymbol{I}_n)\boldsymbol{w}] = [(\boldsymbol{b}^{(t)}\boldsymbol{a}_O) \otimes \boldsymbol{I}_n]\boldsymbol{w}. \end{aligned} \quad (10)$$

4.4 Batch Proving by Operation Type

For each operation type $t \in [T]$, we prove the $M^{(t)}$ gate constraints in one batched equation. We collect the left inputs, right inputs, and outputs of type-t gates into $\boldsymbol{w}_L^{(t)}, \boldsymbol{w}_R^{(t)}, \boldsymbol{w}_O^{(t)} \in \mathbb{F}^{(Mn)\times n}$, with zeros outside the type-t blocks. In Figure 3, the NN has multiplication, addition, and activation gates, and the corresponding block matrices $\{\boldsymbol{w}_{L/R/O}^{(t)}\}_{t\in\{\text{mul},+,\phi\}}$ omit the zero blocks for readability. For each t,

we express the $M^{(t)}$ gate constraints as a single matrix equation over $\boldsymbol{w}_L^{(t)}, \boldsymbol{w}_R^{(t)}, \boldsymbol{w}_O^{(t)}$.

Batched matrix multiplication. For multiplication, correctness of all M^{mul} matrix multiplications is equivalent to the diagonal-block equation (11) below, where $\text{Diag}_n(\cdot)$ rearranges the M blocks of an $(Mn) \times n$ block matrix into an $(Mn) \times (Mn)$ diagonal-block matrix. Figure 2 illustrates this transformation.

$$\text{Diag}_n(\boldsymbol{w}_L^{\text{mul}}) \cdot \text{Diag}_n(\boldsymbol{w}_R^{\text{mul}}) = \text{Diag}_n(\boldsymbol{w}_O^{\text{mul}}) \tag{11}$$

Batched element-wise operations. Element-wise operations, *e.g.*, matrix addition, Hadamard product, and element-wise nonlinear functions ϕ, can be verified collectively as:

$$\boldsymbol{w}_L^{+} + \boldsymbol{w}_R^{+} = \boldsymbol{w}_O^{+}, \quad \boldsymbol{w}_L^{\circ} \circ \boldsymbol{w}_R^{\circ} = \boldsymbol{w}_O^{\circ}, \quad \phi(\boldsymbol{w}_L^{\phi}) = \boldsymbol{w}_O^{\phi}.$$

The nonlinear ϕ is verified via sparse lookup proofs (Appendix C.4).

Batched convolution. Convolution gates are proven using the property (Appendix B.2.5) that $\boldsymbol{c} = \text{Conv}(\boldsymbol{a}, \boldsymbol{b})$ holds iff, for random $y, y' \in \mathbb{F}^*$ and $\vec{\boldsymbol{y}} \coloneqq (1, y, \ldots, y^{n-1})$, $\vec{\boldsymbol{y}'} \coloneqq (1, y', \ldots, (y')^{n-1})$:

$$\vec{\boldsymbol{y}}^{\mathsf{T}}_{[:n_a+n_b]} \boldsymbol{c} \vec{\boldsymbol{y}'}_{[:n_a+n_b]} = (\vec{\boldsymbol{y}}^{\mathsf{T}}_{[:n_a]} \boldsymbol{a} \vec{\boldsymbol{y}'}_{[:n_a]}) \cdot (\vec{\boldsymbol{y}}^{\mathsf{T}}_{[:n_b]} \boldsymbol{b} \vec{\boldsymbol{y}'}_{[:n_b]}). \tag{12}$$

The same $\vec{\boldsymbol{y}}$ and $\vec{\boldsymbol{y}'}$ can be reused for all M^{conv} convolution gates. Under Kronecker lifting, the M^{conv} instances of Eq. (12) merge into:

$$\begin{aligned} &[(\boldsymbol{I}_{M^{\text{conv}}} \otimes \vec{\boldsymbol{y}}^{\mathsf{T}}) \cdot \text{Diag}_n(\boldsymbol{w}_L^{\text{conv}}) \cdot (\boldsymbol{I}_{M^{\text{conv}}} \otimes \vec{\boldsymbol{y}'})] \\ &\cdot [(\boldsymbol{I}_{M^{\text{conv}}} \otimes \vec{\boldsymbol{y}}^{\mathsf{T}}) \cdot \text{Diag}_n(\boldsymbol{w}_R^{\text{conv}}) \cdot (\boldsymbol{I}_{M^{\text{conv}}} \otimes \vec{\boldsymbol{y}'})] \\ &= [(\boldsymbol{I}_{M^{\text{conv}}} \otimes \vec{\boldsymbol{y}}^{\mathsf{T}}) \cdot \text{Diag}_n(\boldsymbol{w}_O^{\text{conv}}) \cdot (\boldsymbol{I}_{M^{\text{conv}}} \otimes \vec{\boldsymbol{y}'})]. \end{aligned} \tag{13}$$

Blocks in $\boldsymbol{w}_{L/R/O}^{\text{conv}}$ are zero-padded as needed to match dimensions.

4.5 Formulating HD-R1CS

By substituting $\{\boldsymbol{w}_L^{(t)}, \boldsymbol{w}_R^{(t)}, \boldsymbol{w}_O^{(t)}\}$ in Eqs. (11)–(13) with Eq. (10), we obtain HD-R1CS. Given commitments to the wiring matrices $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$ and the selector matrices $\{\boldsymbol{b}^{(t)}\}_{t=0}^{T}$, Mat-Circ-SAT reduces to $T+1$ HD-R1CS equations. For each operation type $*^{(t)}$, one equation enforces the type-specific constraint on its left, right, and output values. The corresponding per-type blocks are extracted from $\boldsymbol{w}$ by the selector–alignment matrices $\{\boldsymbol{b}^{(t)}\boldsymbol{a}_L, \boldsymbol{b}^{(t)}\boldsymbol{a}_R, \boldsymbol{b}^{(t)}\boldsymbol{a}_O\}$.

Definition 5 (HD-R1CS). *Let C_L, C_R, C_O, $\{C^{(t)}\}_{t=0}^{T}$, and C_{io} be commitments to $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$, to $\{\boldsymbol{b}^{(t)}\}_{t=0}^{T}$, and to the I/Os, respectively. An* HD-R1CS instance *is a tuple:*

$$\left(\mathbb{F}, n, M, \mathbb{T}, C_L, C_R, C_O, \{C^{(t)}\}_{t=0}^{T}, C_{\text{io}} : \boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O, \{\boldsymbol{b}^{(t)}\}_{t=0}^{T}, \boldsymbol{w}\right).$$

The HD-R1CS relation $\mathcal{R}_{\text{HD-R1CS}}$ consists of $T+1$ equations:

(1) one equation ensuring that C_{io} opens to $(\boldsymbol{b}^{(0)} \otimes \boldsymbol{I}_n)\boldsymbol{w}$, which extracts the I/O blocks; and

(2) T operation-specific equations (Eqs. (11)–(13)*), one for each $t \in [T]$. The t-th equation enforces the operation-t constraint on $\{\boldsymbol{w}_L^{(t)}, \boldsymbol{w}_R^{(t)}, \boldsymbol{w}_O^{(t)}\}$, where:*

$$\begin{aligned} \boldsymbol{w}_L^{(t)} &= [(\boldsymbol{b}^{(t)}\boldsymbol{a}_L) \otimes \boldsymbol{I}_n]\boldsymbol{w}, \qquad \boldsymbol{w}_R^{(t)} = [(\boldsymbol{b}^{(t)}\boldsymbol{a}_R) \otimes \boldsymbol{I}_n]\boldsymbol{w}, \\ \boldsymbol{w}_O^{(t)} &= [(\boldsymbol{b}^{(t)}\boldsymbol{a}_O) \otimes \boldsymbol{I}_n]\boldsymbol{w}, \qquad t \in [T]. \end{aligned}$$

For instance, the HD-R1CS equation for matrix multiplication is:

$$\begin{aligned} &\text{Diag}_n\Big([(\boldsymbol{b}^{\text{mul}}\boldsymbol{a}_L) \otimes \boldsymbol{I}_n]\boldsymbol{w}\Big) \cdot \text{Diag}_n\Big([(\boldsymbol{b}^{\text{mul}}\boldsymbol{a}_R) \otimes \boldsymbol{I}_n]\boldsymbol{w}\Big) \\ &= \text{Diag}_n\Big([(\boldsymbol{b}^{\text{mul}}\boldsymbol{a}_O) \otimes \boldsymbol{I}_n]\boldsymbol{w}\Big), \end{aligned} \tag{14}$$

with analogous equations for other operation types.

Why these matrices are structured. All large matrices in the HD-R1CS equations are formed by Kronecker lifting with $\boldsymbol{I}_n$ and by diagonal-block rearrangements via $\text{Diag}_n(\cdot)$. This Kronecker-and-block structure is what enables our structure-aware proofs.

5 Building Blocks: Structure-Aware Proofs

Structured sparsity in HD-R1CS. Kronecker lifting and diagonal-block transformations in HD-R1CS give rise to large sparse matrices of dimension $(Mn) \times (Mn)$. This appears problematic, since committing to and opening large sparse matrices is a major bottleneck in monolithic-circuit approaches to zkML. HD-R1CS does not avoid large matrices syntactically, but it restricts them to structured families that admit efficient MLE reductions. As shown in Table 2, large matrices in the HD-R1CS equations fall into two structured classes.

Class ① consists of operation-t-specific rotation matrices, *i.e.*, $(\boldsymbol{b}^{(t)}\boldsymbol{a}_{L/R/O}) \otimes \boldsymbol{I}_n$, with Kronecker structure.

Class ② consists of diagonal-block matrices $\text{Diag}_n(\boldsymbol{w}_{L/R/O}^{(t)})$.

Both classes admit efficient MLE reductions by factoring them into smaller components. Unstructured sparse matrices have been extensively studied in the context of R1CS. Prior work on sparse arithmetization, notably Marlin [22] and Spartan [53] (see Appendix B.3), addresses general sparsity but incurs substantial overhead. In contrast, our approach exploits the structured sparsity inherent in HD-R1CS, enabling more efficient structure-aware proofs.

We work within the PIOP+PCS paradigm, where verifying a matrix relation reduces to checking a polynomial relation over the MLEs of the input matrices, and the required MLE evaluations are proven with a PCS. Theorem 1 below is standard in the literature.

Theorem 1. *[22, Theorem 8.4, reformulated] Let Π be a PIOP with negligible soundness error that reduces a relation $\mathcal{R}$ to the verification of polynomial evaluations, and let* PCS *be a polynomial commitment scheme as in Definition 2 that satisfies knowledge soundness and hiding. If the polynomial evaluations reduced from Π are verified via* PCS*, then the resulting protocol is a zkSNARK for $\mathcal{R}$.*

Our focus is on the PIOP layer. Concretely, each building block below reduces an MLE evaluation involving a large structured matrix to a small number of MLE evaluations of smaller matrices and vectors. These reduced evaluations are then discharged by invoking the PCS opening protocol for the corresponding commitments.

Beyond the standard PIOPs in Appendix B.2, the structured sparsity of HD-R1CS motivates more efficient PIOPs for large structured matrices. These PIOPs operate blockwise on block matrices. Accordingly, we develop structure-aware proofs for the following.

- *MLE reduction for Kronecker products, e.g.,* $(\boldsymbol{b}^{(t)}\boldsymbol{a}_{L/R/O}) \otimes \boldsymbol{I}_n$.
- *MLE reduction for diagonal-block matrices, e.g.,* $\text{Diag}_n(\boldsymbol{w}_L^{(t)})$.

Table 2: Dimension and sparsity of matrices in HD-R1CS

Variable	Symbol	Dimension	# Non-Zero
Witness	$\boldsymbol{w}$	$(nM) \times n$	n^2M
Packed gate IOs	$\boldsymbol{w}_L, \boldsymbol{w}_R, \boldsymbol{w}_O$	$(nM) \times n$	n^2M
Gate IOs for op. t	$\boldsymbol{w}_L^{(t)}, \boldsymbol{w}_R^{(t)}, \boldsymbol{w}_O^{(t)}$	$(nM) \times n$	$n^2M^{(t)}$
Circuit specification	$\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O$	$M \times M$	$\le M$
Op.-t-specific selection	$\boldsymbol{b}^{(t)}, t \in [T]$	$M \times M$	$M^{(t)}$
Op.-t-specific rotation	$(\boldsymbol{b}^{(t)}\boldsymbol{a}_{L/R/O}) \otimes \boldsymbol{I}_n$	$(nM) \times (nM)$	$nM^{(t)}$
Diagonal-block gate IOs	$\mathrm{Diag}_n(\boldsymbol{w}_{L/R/O}^{(t)})$	$(nM) \times (nM)$	$n^2M^{(t)}$

Algorithm 1: Proof for $v = \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r)$, $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$

Input: Public: $v \in \mathbb{F}, \boldsymbol{x}_l \in \mathbb{F}^{\log_2(m_a m_b)}, \boldsymbol{x}_r \in \mathbb{F}^{\log_2(n_a n_b)}$
Private: $\boldsymbol{a} \in \mathbb{F}^{m_a \times n_a}, \boldsymbol{b} \in \mathbb{F}^{m_b \times n_b}, \boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$
Output: true or false

1 $\mathcal{P}$ computes: $v_a \leftarrow \widehat{\boldsymbol{a}}(\boldsymbol{x}_{al}, \boldsymbol{x}_{ar})$ and $v_b \leftarrow \widehat{\boldsymbol{b}}(\boldsymbol{x}_{bl}, \boldsymbol{x}_{br})$ where $(\boldsymbol{x}_{al}, \boldsymbol{x}_{bl}) = \boldsymbol{x}_l$ and $(\boldsymbol{x}_{ar}, \boldsymbol{x}_{br}) = \boldsymbol{x}_r$;
2 $\mathcal{P} \to \mathcal{V}$: $v_a, v_b \in \mathbb{F}$;
3 $\mathcal{V}$: **if** $v \neq v_a \cdot v_b$ **then return** false;
4 $\mathcal{P}$ produces additional proofs to let $\mathcal{V}$ verify $v_a = \widehat{\boldsymbol{a}}(\boldsymbol{x}_{al}, \boldsymbol{x}_{ar})$ and $v_b = \widehat{\boldsymbol{b}}(\boldsymbol{x}_{bl}, \boldsymbol{x}_{br})$, where $\boldsymbol{x}_l = (\boldsymbol{x}_{al}, \boldsymbol{x}_{bl})$ and $\boldsymbol{x}_r = (\boldsymbol{x}_{ar}, \boldsymbol{x}_{br})$;

- *Blockwise matrix multiplication, e.g.*, multiplying $[(\boldsymbol{b}^{(t)}\boldsymbol{a}_L) \otimes \boldsymbol{I}_n]$ with $\boldsymbol{w}$, and multiplying $\mathrm{Diag}_n(\boldsymbol{w}_L^{\mathrm{mul}})$ with $\mathrm{Diag}_n(\boldsymbol{w}_R^{\mathrm{mul}})$.

5.1 Kronecker MLE Reduction

For Kronecker-structured matrices in HD-R1CS, the MLE of a Kronecker product factors as the product of the MLEs of its factors. Proposition 1 yields Algorithm 1, reducing $v = \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r)$ to MLE evaluations of the factors. See Appendix D.1 for proof.

Proposition 1. *For $\boldsymbol{a} \in \mathbb{F}^{m_a \times n_a}$, $\boldsymbol{b} \in \mathbb{F}^{m_b \times n_b}$, and $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$:*

$$\widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \widehat{\boldsymbol{a}}(\boldsymbol{x}_{al}, \boldsymbol{x}_{ar}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{x}_{bl}, \boldsymbol{x}_{br}),$$

where $\boldsymbol{x}_l = (\boldsymbol{x}_{al}, \boldsymbol{x}_{bl})$ and $\boldsymbol{x}_r = (\boldsymbol{x}_{ar}, \boldsymbol{x}_{br})$.

5.2 Diagonal-Block MLE Reduction

The MLE of a diagonal-block matrix admits a structured decomposition that allows efficient evaluation via a single MLE evaluation of the underlying dense block matrix.

Proposition 2. *Let $\boldsymbol{c} = \mathrm{Diag}_n(\boldsymbol{a})$, where $\boldsymbol{a} \in \mathbb{F}^{(Mn) \times n}$ consists of M blocks of size $n \times n$. Assume $1 - x'_{lj} \neq 0$ for all $j \in [\log_2 M]$. Then:*

$$\widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \left[\prod_{j=1}^{\log_2 M} \frac{(1 - x_{alj})(1 - x_{arj})}{1 - x'_{lj}}\right] \cdot \widehat{\boldsymbol{a}}\big((\boldsymbol{x}'_l, \boldsymbol{x}_{bl}), \boldsymbol{x}_{br}\big), \quad (15)$$

where $\boldsymbol{c} \in \mathbb{F}^{(Mn) \times (Mn)}$, $\boldsymbol{x}_l = (\boldsymbol{x}_{al}, \boldsymbol{x}_{bl}) \in \mathbb{F}^{\log_2(Mn)}$, and $\boldsymbol{x}_r = (\boldsymbol{x}_{ar}, \boldsymbol{x}_{br}) \in \mathbb{F}^{\log_2(Mn)}$. The point $\boldsymbol{x}'_l \in \mathbb{F}^{\log_2 M}$ is determined by:

$$\boldsymbol{x}'_l / (\boldsymbol{1} - \boldsymbol{x}'_l) = (\boldsymbol{x}_{al} \circ \boldsymbol{x}_{ar}) / [(\boldsymbol{1} - \boldsymbol{x}_{al}) \circ (\boldsymbol{1} - \boldsymbol{x}_{ar})]. \quad (16)$$

The proof is given in Appendix D.2. Proposition 2 yields Algorithm 2 for proving the MLE evaluation of a diagonal-block matrix.

Algorithm 2: Proof for $v = \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r)$, $\boldsymbol{c} = \mathrm{Diag}_n(\boldsymbol{a})$

Input: Public: $v \in \mathbb{F}, \boldsymbol{x}_l \in \mathbb{F}^{\log_2(Mn)}, \boldsymbol{x}_r \in \mathbb{F}^{\log_2(Mn)}$
Private: $\boldsymbol{a} \in \mathbb{F}^{(Mn) \times n}, \boldsymbol{c} = \mathrm{Diag}_n(\boldsymbol{a})$
Output: true or false

1 $\mathcal{P}$ computes: $v_a \leftarrow \widehat{\boldsymbol{a}}((\boldsymbol{x}'_l, \boldsymbol{x}_{bl}), \boldsymbol{x}_{br})$ where $(\boldsymbol{x}_{al}, \boldsymbol{x}_{bl}) = \boldsymbol{x}_l$, $(\boldsymbol{x}_{ar}, \boldsymbol{x}_{br}) = \boldsymbol{x}_r$, and $\boldsymbol{x}'_l$ is determined by Eq. (16);
2 $\mathcal{P} \to \mathcal{V}$: $v_a \in \mathbb{F}$;
3 $\mathcal{V}$: Set $\boldsymbol{x}_l = (\boldsymbol{x}_{al}, \boldsymbol{x}_{bl})$ and $\boldsymbol{x}_r = (\boldsymbol{x}_{ar}, \boldsymbol{x}_{br})$, derive $\boldsymbol{x}'_l$ from Eq. (16), and set $\hat{v} \leftarrow \left(\prod_{j=1}^{\log_2 M} \frac{(1-x_{alj})(1-x_{arj})}{1-x'_{lj}}\right) \cdot v_a$;
4 $\mathcal{V}$: **if** $v \neq \hat{v}$ **then return** false;
5 $\mathcal{P}$ produces additional proofs to let $\mathcal{V}$ verify $v_a = \widehat{\boldsymbol{a}}((\boldsymbol{x}'_l, \boldsymbol{x}_{bl}), \boldsymbol{x}_{br})$;

5.3 Blockwise Matrix Multiplication Proof

The HD-R1CS equations in Eqs. (11) and (13) require proving large matrix multiplications. All such matrices admit a block structure, allowing the proofs to be carried out blockwise.

Thaler's matrix multiplication proof. Thaler [59] proposed using sumcheck (Appendix B.2) to verify $\boldsymbol{c} = \boldsymbol{ab} \in \mathbb{F}^{n \times n}$ at a challenge point $(\boldsymbol{x}_l, \boldsymbol{x}_r) \in \mathbb{F}^{2\log_2 n}$:

$$\widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \sum_{\boldsymbol{x} \in \{0,1\}^{\log_2 n}} \widehat{\boldsymbol{a}}(\boldsymbol{x}_l, \boldsymbol{x}) \widehat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{x}_r).$$

Its primary cost is computing MLE evaluations derived from $\boldsymbol{a}$ and $\boldsymbol{b}$ under the verifier challenges.

MLE evaluation via projections. By Lemma 1, MLE evaluations can be expressed as matrix projections. Specifically:

$$\widehat{\boldsymbol{a}}(\boldsymbol{x}_l, \boldsymbol{x}) = \widehat{(\vec{\boldsymbol{l}}^{\top}\boldsymbol{a})}(\boldsymbol{x}), \qquad \widehat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{x}_r) = \widehat{(\boldsymbol{b}\vec{\boldsymbol{r}})}(\boldsymbol{x}),$$

where $\vec{\boldsymbol{l}}$ and $\vec{\boldsymbol{r}}$ are tensor-structured vectors derived from $\boldsymbol{x}_l$ and $\boldsymbol{x}_r$ as in Eq. (6). Thus, verifying matrix multiplication reduces to computing and proving single-sided projections of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Blockwise projection. Matrix projections decompose along block boundaries. Dense-block and diagonal-block matrices therefore admit efficient blockwise projection algorithms. Write a tensor-structured vector as $\vec{\boldsymbol{l}} = \vec{\boldsymbol{l}}^{(M)} \otimes \vec{\boldsymbol{l}}^{(n)}$, where $\vec{\boldsymbol{l}}^{(M)} \in \mathbb{F}^M$ and $\vec{\boldsymbol{l}}^{(n)} \in \mathbb{F}^n$, and analogously for $\vec{\boldsymbol{r}}$.

Let $\boldsymbol{w} \in \mathbb{F}^{(Mn) \times n}$ be a dense block matrix consisting of M blocks $\{\boldsymbol{w}_i\}_{i=1}^M$, each of size $n \times n$. For $\vec{\boldsymbol{l}} = \vec{\boldsymbol{l}}^{(M)} \otimes \vec{\boldsymbol{l}}^{(n)}$ and $\vec{\boldsymbol{r}} \in \mathbb{F}^n$, the projection $\vec{\boldsymbol{l}}^{\top}\boldsymbol{w}\vec{\boldsymbol{r}}$ can be computed block-by-block as:

$$\vec{\boldsymbol{l}}^{\top}\boldsymbol{w}\vec{\boldsymbol{r}} = \sum_{i=1}^{M} \big(\vec{\boldsymbol{l}}_i^{(M)}\big) \cdot \big(\vec{\boldsymbol{l}}^{(n)}\big)^{\top} \boldsymbol{w}_i \vec{\boldsymbol{r}}. \quad (17)$$

For a diagonal-block matrix $\boldsymbol{W} \in \mathbb{F}^{(Mn) \times (Mn)}$ with M blocks $\{\boldsymbol{W}_i\}_{i=1}^M$ of size $n \times n$, and tensor-structured vectors $\vec{\boldsymbol{l}} = \vec{\boldsymbol{l}}^{(M)} \otimes \vec{\boldsymbol{l}}^{(n)}$ and $\vec{\boldsymbol{r}} = \vec{\boldsymbol{r}}^{(M)} \otimes \vec{\boldsymbol{r}}^{(n)}$, its projection decomposes as:

$$\vec{\boldsymbol{l}}^{\top}\boldsymbol{W}\vec{\boldsymbol{r}} = \sum_{i=1}^{M} \big(\vec{\boldsymbol{l}}_i^{(M)}\vec{\boldsymbol{r}}_i^{(M)}\big) \cdot \big(\vec{\boldsymbol{l}}^{(n)}\big)^{\top} \boldsymbol{W}_i \vec{\boldsymbol{r}}^{(n)}.$$

In these blockwise computations, padded zero blocks never contribute and can be omitted.

Algorithm 3: Blockwise matrix multiplication proof

Input: Private: $\mathrm{Diag}_n(\boldsymbol{a}), \mathrm{Diag}_n(\boldsymbol{b}), \mathrm{Diag}_n(\boldsymbol{c}) \in \mathbb{F}^{(Mn)\times(Mn)}$, where $\mathrm{Diag}_n(\boldsymbol{c}) = \mathrm{Diag}_n(\boldsymbol{a}) \cdot \mathrm{Diag}_n(\boldsymbol{b})$

Output: true or false

$\mathcal{V} \to \mathcal{P}$: $\boldsymbol{x}_l, \boldsymbol{x}_r \in \mathbb{F}^{\log_2 M}$; $\boldsymbol{x}'_l, \boldsymbol{x}'_r \in \mathbb{F}^{\log_2 n}$;

$\mathcal{P}$ computes:

$\vec{\boldsymbol{l}} \leftarrow (1-x_{l1}, x_{l1}) \otimes \cdots \otimes (1-x_{l\log_2 M}, x_{l\log_2 M}) \in \mathbb{F}^M$;

$\vec{\boldsymbol{r}} \leftarrow (1-x_{r1}, x_{r1}) \otimes \cdots \otimes (1-x_{r\log_2 M}, x_{r\log_2 M}) \in \mathbb{F}^M$;

$\vec{\boldsymbol{l}'} \leftarrow (1-x'_{l1}, x'_{l1}) \otimes \cdots \otimes (1-x'_{l\log_2 n}, x'_{l\log_2 n}) \in \mathbb{F}^n$;

$\vec{\boldsymbol{r}'} \leftarrow (1-x'_{r1}, x'_{r1}) \otimes \cdots \otimes (1-x'_{r\log_2 n}, x'_{r\log_2 n}) \in \mathbb{F}^n$;

For each block $\boldsymbol{c}_i \in \mathbb{F}^{n\times n}$, compute $\widehat{c}_i \leftarrow (\vec{\boldsymbol{l}'})^{\mathsf{T}} \boldsymbol{c}_i \vec{\boldsymbol{r}'} \in \mathbb{F}$;

Aggregate $v_c \leftarrow \sum_{i=1}^{M} (\vec{\boldsymbol{l}}_i \vec{\boldsymbol{r}}_i)\, \widehat{c}_i \in \mathbb{F}$;

For each block $\boldsymbol{a}_i \in \mathbb{F}^{n\times n}$, compute $\vec{\boldsymbol{a}}_i \leftarrow (\vec{\boldsymbol{l}})^{\mathsf{T}} \boldsymbol{a}_i \in \mathbb{F}^n$;

Assemble $\vec{\boldsymbol{a}}_l \leftarrow (\vec{\boldsymbol{l}}_1 \vec{\boldsymbol{a}}_1 \,\|\, \cdots \,\|\, \vec{\boldsymbol{l}}_M \vec{\boldsymbol{a}}_M) \in \mathbb{F}^{Mn}$;

For each block $\boldsymbol{b}_i \in \mathbb{F}^{n\times n}$, compute $\vec{\boldsymbol{b}}_i \leftarrow \boldsymbol{b}_i \vec{\boldsymbol{r}'} \in \mathbb{F}^n$;

Assemble $\vec{\boldsymbol{b}}_r \leftarrow (\vec{\boldsymbol{r}}_1 \vec{\boldsymbol{b}}_1 \,\|\, \cdots \,\|\, \vec{\boldsymbol{r}}_M \vec{\boldsymbol{b}}_M) \in \mathbb{F}^{Mn}$;

$\mathcal{P} \to \mathcal{V}$: $v_c \in \mathbb{F}$;

Prover and verifier run a sumcheck protocol for $v_c = \sum_{\boldsymbol{x}\in\{0,1\}^{\log_2(Mn)}} \widehat{\vec{\boldsymbol{a}}_l}(\boldsymbol{x}) \cdot \widehat{\vec{\boldsymbol{b}}_r}(\boldsymbol{x})$, which reduces verification to checking $v_a = \widehat{\vec{\boldsymbol{a}}_l}(\boldsymbol{x})$ and $v_b = \widehat{\vec{\boldsymbol{b}}_r}(\boldsymbol{x})$ at a random point $\boldsymbol{x}$;

$\mathcal{P}$ provides additional proofs to let $\mathcal{V}$ verify the required MLE evaluations of the committed diagonal-block multiplicands that imply v_a and v_b;

Blockwise matrix multiplication proof. Combining blockwise projection with Thaler's matrix multiplication protocol yields efficient multiplication proofs for large block matrices. Algorithm 3 instantiates this approach for diagonal-block multiplication as in Eq. (11). For diagonal-block matrices with $M^{(t)}$ nonzero blocks of size $n \times n$, the prover time is $O(M^{(t)} n^2)$.

Handling heterogeneous block dimensions. The preceding discussion assumes blocks have a uniform shape $n \times n$. To accommodate heterogeneous matrix dimensions, we conceptually zero-pad each matrix to this common shape. These padded zeros are never materialized under blockwise computation and therefore incur no significant memory overhead.

Concretely, consider the witness matrix $\boldsymbol{w} \in \mathbb{F}^{Mn\times n}$ consisting of M blocks, each padded to size $n \times n$. Rather than materializing padded matrices, we represent $\boldsymbol{w}$ as a collection of blocks together with their true dimensions: $\{\boldsymbol{w}_i,\ (m_i, n_i)\}_{i\in[M]}$. This representation avoids storing padded zeros.

All blockwise computations in HD-R1CS are compatible with this representation. Given block dimensions, the projection in Eq. (17) can be computed by accessing only the unpadded portions of each block. HD-R1CS supports heterogeneous block sizes without instantiating padded zeros. It incurs negligible asymptotic overhead in memory and prover time.

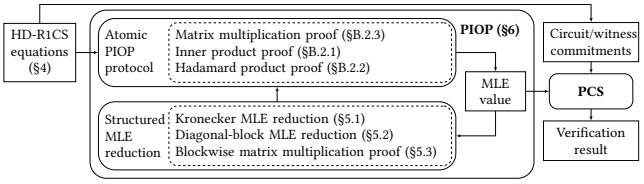


Figure 4: Flowchart of zkSMART in the PIOP+PCS paradigm.

6 zkSMART: zkSNARK for HD-R1CS

We prove the HD-R1CS constraints in Eqs. (11)–(13) in the PIOP+PCS paradigm. The resulting compiler is zkSMART. It reduces each HD-R1CS matrix equation to a bounded set of MLE claims and checks them via PCS openings. By the composition theorem in Section 5, this yields a zkSNARK for HD-R1CS when the PCS is knowledge-sound and hiding.

We view each HD-R1CS equation as a matrix-operation graph. zkSMART starts from the root MLE claim and applies an atomic PIOP at each node to reduce it to a small set of child MLE claims at related points. For linear-algebraic operations we use sumcheck-based PIOPs, and for nonlinearities we use sparse lookup proofs. The recursion ends with PCS openings of $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$, $\{\boldsymbol{b}^{(t)}\}_{t=1}^{T}$, and $\boldsymbol{w}$. (See Appendix B.2 and Appendix C.4 for the concrete atomic protocols and their reductions to PCS openings.)

During the reduction, the protocol encounters large structured matrices that are either Kronecker-structured or block-structured. For these matrices, we apply the MLE reductions and blockwise proofs from Section 5 to avoid generic sparse-matrix overhead. Concretely, the Kronecker MLE reduction in Section 5.1 and the diagonal-block MLE reduction in Section 5.2 replace one large MLE evaluation with a small number of MLE evaluations on smaller factors. The blockwise matrix multiplication proof in Section 5.3 replaces monolithic projections with blockwise projections that skip zero-padded blocks. These structure-aware steps preserve the PIOP+PCS reduction pattern while improving prover efficiency.

The zkSMART protocol in Algorithm 4 proceeds as follows:

- *Offline commitment*. Commit to $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$, $\{\boldsymbol{b}^{(t)}\}_{t=0}^{T}$, and the I/O matrices (*i.e.*, NN parameters, inputs, and outputs).
- *Online commitment*. Commit to the witness matrix $\boldsymbol{w}$.
- *PIOP reduction*. Recursively apply the atomic PIOPs from Appendix B.2 and Appendix C.4. Apply the MLE reductions from Section 5.1 and Section 5.2 whenever a Kronecker or diagonal-block matrix arises. Apply the blockwise matrix multiplication proof from Section 5.3 whenever a large block multiplication arises. This reduces each HD-R1CS equation to MLE evaluations of $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$, $\{\boldsymbol{b}^{(t)}\}_{t=1}^{T}$, and $\boldsymbol{w}$ at random points.
- *PCS opening*. Use the Open algorithm of the PCS to verify the reduced MLE evaluation claims on $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O\}$, $\{\boldsymbol{b}^{(t)}\}_{t=1}^{T}$, and $\boldsymbol{w}$.

Consider Eq. (14), the multiplication constraint:

$$\mathrm{Diag}_n([(\boldsymbol{b}^{\mathsf{mul}} \boldsymbol{a}_L) \otimes \boldsymbol{I}_n]\boldsymbol{w}) \cdot \mathrm{Diag}_n([(\boldsymbol{b}^{\mathsf{mul}} \boldsymbol{a}_R) \otimes \boldsymbol{I}_n]\boldsymbol{w}) = \mathrm{Diag}_n([(\boldsymbol{b}^{\mathsf{mul}} \boldsymbol{a}_O) \otimes \boldsymbol{I}_n]\boldsymbol{w}).$$

Algorithm 3 reduces the above to three MLE claims on diagonal-block matrices at a verifier-chosen point. Section 5.2 reduces each

Algorithm 4: zkSMART Protocol for HD-R1CS

Input: Public: field $\mathbb{F}$; operation set $\mathbb{T}$; sizes $n, M \in \mathbb{N}$; commitments $C_L, C_R, C_O, \{C^{(t)}\}_{t=0}^{|\mathbb{T}|}, C_{\text{io}} \in \mathbb{C}$
Private: sparse matrices $\{\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O \in \mathbb{F}^{M\times M}\}$, $\{\boldsymbol{b}^{(t)} \in \mathbb{F}^{M\times M}\}_{t=0}^{|\mathbb{T}|}$; witness matrix $\boldsymbol{w} \in \mathbb{F}^{(Mn)\times n}$

Output: true or false

Online commitment phase:

$\mathcal{P}$ commits to the witness matrix $\boldsymbol{w}$, obtaining $C_w \in \mathbb{C}$;

$\mathcal{P} \to \mathcal{V}$: send C_w;

Proof reduction phase:

For each of the $|\mathbb{T}| + 1$ HD-R1CS equations (I/O constraint and one per operation type):

$\mathcal{P}$ and $\mathcal{V}$ execute the corresponding PIOP as follows:

- recursive sumcheck for each matrix operation in HD-R1CS;
- blockwise matrix multiplication proof where applicable;
- MLE reductions for Kronecker and diagonal-block matrices;

The PIOPs reduce verification to checking MLE evaluations of $\boldsymbol{a}_L, \boldsymbol{a}_R, \boldsymbol{a}_O, \{\boldsymbol{b}^{(t)}\}_{t=0}^{|\mathbb{T}|}$, and $\boldsymbol{w}$;

Opening phase:

$\mathcal{P}$ and $\mathcal{V}$ run the PCS opening algorithm to verify all required MLE evaluations;

return true if all checks succeed; otherwise return false.

diagonal-block claim to an MLE claim on the corresponding dense block matrix at a related point. Section 5.3 then reduces each dense-block claim to MLE claims on $(\boldsymbol{b}^{\text{mul}}\boldsymbol{a}_{L/R/O}) \otimes \boldsymbol{I}_n$ and on $\boldsymbol{w}$. Section 5.1 reduces the Kronecker term to MLE claims on $\boldsymbol{b}^{\text{mul}}\boldsymbol{a}_{L/R/O}$, while $\widehat{\boldsymbol{I}_n}$ is computed by the verifier. Finally, standard multiplication PIOPs reduce $\boldsymbol{b}^{\text{mul}}\boldsymbol{a}_{L/R/O}$ to MLE claims on $\boldsymbol{b}^{\text{mul}}$ and $\boldsymbol{a}_{L/R/O}$. All leaf claims on $\boldsymbol{a}_{L/R/O}$, $\boldsymbol{b}^{\text{mul}}$, and $\boldsymbol{w}$ are verified via PCS openings.

HD-R1CS equations for other operations follow the same reduction pattern. For example, element-wise activation functions ϕ are handled by the lookup proof in Appendix C.4, which reduces the HD-R1CS equation to MLE evaluations of $[(\boldsymbol{b}^{(t)}\boldsymbol{a}_{L/O}) \otimes \boldsymbol{I}_n]\boldsymbol{w}$. After this step, the remaining reductions and PCS openings follow the same template as in the multiplication case.

Complexity. We treat T as a constant determined by the fixed operation set $\mathbb{T}$. Blockwise matrix multiplication for the type-mul constraints takes $O(n^2M^{\text{mul}})$ prover time. Across the $T + 1$ HD-R1CS equations, the PIOP layer takes $O(n^2M)$ prover time, and polynomial commitments and openings also take $O(n^2M)$ prover time. Overall, the prover time is $O(n^2M)$. The prover's RAM usage is dominated by storing $\boldsymbol{w}$ and the auxiliary per-type matrices $\{\boldsymbol{w}_L^{(t)}, \boldsymbol{w}_R^{(t)}, \boldsymbol{w}_O^{(t)}\}$, for total space $O(n^2M)$. The proof size and verifier time are both $O(\log(nM))$ under the same constant-T assumption, since each invoked subprotocol has logarithmic cost and is used a constant number of times per equation.

7 Quantization for HD-R1CS

Quantization as an encoding choice. We prove correctness of a fixed quantized semantics, not of real-valued floating-point arithmetic. We support low-bitwidth integer arithmetic sufficient to encode common NN inference computations in HD-R1CS. Our protocol proves correctness of the specified quantized computation. We treat the quantization scheme and its accuracy impact as external, and we make no accuracy claims. We use standard 8-bit quantization primitives (scaling, rounding, saturation) [71] and encode them in HD-R1CS. We assume $\mathbb{F}$ is large enough to represent all intermediate integers enforced by the quantization constraints.

Quantization for NN. To prove floating-point NN computation, zkML systems typically fix a quantization scheme and then prove the induced integer computation over $\mathbb{F}$. Prior zkML work uses either bit-decomposition-based methods (*e.g.*, [67]) or scaling-factor-based methods (*e.g.*, [42, 47, 57]), and both induce approximation error relative to real arithmetic. We do not propose a new quantization method. We adopt standard quantized-NN practice from the ML community [41, 71]. In industrial deployments, NN inference is routinely performed in low-precision integer arithmetic (most commonly 8-bit integers) together with quantization-aware training (*e.g.*, [71]) to control inference error. A joint NVIDIA-Arm-Intel whitepaper summarizes this trend: "For inference, fixed-point INT8 representation is a popular option." [50, Section 1].

In quantized NNs, each floating-point matrix is represented as an integer matrix together with a shared scaling factor. Rounding and overflow are handled by saturating values to a fixed numerical range. Concretely, for two $n \times n$ matrices of 8-bit integers $\boldsymbol{a}$ and $\boldsymbol{b}$, and floating-point scaling factors $\epsilon_a^{\text{f}}, \epsilon_b^{\text{f}}, \epsilon_c^{\text{f}}$, the product $\boldsymbol{c}$ is:

$$\boldsymbol{c} = \min\Big(\max\big(\lfloor(\epsilon_a^{\text{f}}\epsilon_b^{\text{f}}/\epsilon_c^{\text{f}}) \cdot \boldsymbol{ab}\rfloor, -127 \cdot \mathbf{1}\big), 127 \cdot \mathbf{1}\Big),$$

where $\mathbf{1}$ is an $n \times n$ all-ones matrix. The floor $\lfloor\cdot\rfloor$, minimum, and maximum operators are applied element-wise.

Some prior discussions highlight a mismatch between field division and floating-point division in quantized inference [27, 57, p. 4]. In our setting, the ratio $(\epsilon_a^{\text{f}}\epsilon_b^{\text{f}}/\epsilon_c^{\text{f}})$ is compiled into an integer scaling parameter and then treated as part of the quantized instance. This keeps the large computation in integer matrix operations, while confining scaling-factor arithmetic to low-dimensional parameters that integrate cleanly with HD-R1CS.

Structured Quantization. In 8-bit quantized NNs, for example, Q8BERT [71], each floating-point matrix $\boldsymbol{a}^{\text{f}} \in \mathbb{R}^{n\times n}$ is represented as $\boldsymbol{a}^{\text{f}} = \epsilon_a^{\text{f}} \cdot \boldsymbol{a}$. Here, $\boldsymbol{a} \in \mathbb{F}^{n\times n}$ is an 8-bit integer matrix and $\epsilon_a^{\text{f}} \in \mathbb{R}$ is a shared scaling factor. A single scaling factor is applied uniformly to all entries of the matrix.

Let $\boldsymbol{a}^{\text{f}}, \boldsymbol{b}^{\text{f}}, \boldsymbol{c}^{\text{f}}$ have scaling factors $\epsilon_a^{\text{f}}, \epsilon_b^{\text{f}}, \epsilon_c^{\text{f}}$ and corresponding integer matrices $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$. That is, $\boldsymbol{a}^{\text{f}} = \epsilon_a^{\text{f}} \cdot \boldsymbol{a}$, $\boldsymbol{b}^{\text{f}} = \epsilon_b^{\text{f}} \cdot \boldsymbol{b}$, and $\boldsymbol{c}^{\text{f}} = \epsilon_c^{\text{f}} \cdot \boldsymbol{c}$. Via Kronecker lifting, these can be expressed compactly as:

$$\begin{pmatrix}\boldsymbol{a}^{\text{f}}\\ \boldsymbol{b}^{\text{f}}\\ \boldsymbol{c}^{\text{f}}\end{pmatrix} = \left[\begin{pmatrix}\epsilon_a^{\text{f}}\\ \epsilon_b^{\text{f}}\\ \epsilon_c^{\text{f}}\end{pmatrix} \otimes \mathbf{1}_n\right] \circ \begin{pmatrix}\boldsymbol{a}\\ \boldsymbol{b}\\ \boldsymbol{c}\end{pmatrix}.$$

This Kronecker structure confines all floating-point scaling information to the low-dimensional vector $\vec{\boldsymbol{\epsilon}} \coloneqq (\epsilon_a^{\text{f}}, \epsilon_b^{\text{f}}, \epsilon_c^{\text{f}})^{\top}$. As a result, scaling parameters can be handled separately from the large integer matrix operations, enabling efficient proving in HD-R1CS.

Error-proving for rounding. Quantized NNs avoid wrap-around in modular arithmetic by explicitly enforcing range bounds. We model saturation by Clamp_k, which replaces out-of-range values

with the nearest boundary:

$$\mathsf{Clamp}_k(\boldsymbol{a}) \coloneqq \max(\min(\boldsymbol{a}, 2^{k-1}-1),\ -2^{k-1}+1).$$

All operators in Clamp_k are applied element-wise.

Quantized NNs compute a floating-point matrix operation $*$ (multiplication, inner/Hadamard product, and convolution) via a scaled integer operation followed by rounding and saturation:

$$\boldsymbol{c}^{\mathsf{f}} = \boldsymbol{a}^{\mathsf{f}} * \boldsymbol{b}^{\mathsf{f}} \iff \boldsymbol{c} = \mathsf{Clamp}_k\left(\lfloor(\epsilon_a^{\mathsf{f}} \cdot \epsilon_b^{\mathsf{f}}/\epsilon_c^{\mathsf{f}}) \cdot (\boldsymbol{a} * \boldsymbol{b})\rfloor\right), \tag{18}$$

where $\lfloor\cdot\rfloor$ truncates the floating-point fractions.

We implement the scaling-and-floor step in fixed-point form. Define the integer scaling parameter:

$$\theta \coloneqq \left\lfloor 2^{2k} \cdot (\epsilon_a^{\mathsf{f}} \cdot \epsilon_b^{\mathsf{f}})/\epsilon_c^{\mathsf{f}} \right\rfloor.$$

Rescaling is then implemented by applying Trunc_{2k}, which discards the lower $2k$ fractional bits. Under the operand-range assumptions enforced by the quantization constraints, Eq. (19) realizes the rounding in Eq. (18). The Clamp_k and Trunc_{2k} steps are proven using error-proving techniques (see Appendix C.1).

$$\boldsymbol{c} = \mathsf{Clamp}_k\left(\mathsf{Trunc}_{2k}\left(\theta \cdot (\boldsymbol{a} * \boldsymbol{b})\right)\right). \tag{19}$$

Quantized HD-R1CS. The quantized relation in Eq. (19) extends to the entire matrix circuit. All quantized matrix operation gates can be aggregated into a few quantized HD-R1CS constraints.

Consider a matrix circuit with M multiplication gates. For each gate $j \in [M]$, let $\epsilon_{Lj}^{\mathsf{f}}$, $\epsilon_{Rj}^{\mathsf{f}}$, and $\epsilon_{Oj}^{\mathsf{f}}$ denote the floating-point scaling factors of the left input, right input, and output, respectively. We collect these into vectors $\vec{\boldsymbol{\epsilon}}_L^{\mathsf{f}} = \{\epsilon_{Lj}^{\mathsf{f}}\}_{j=1}^{M}$, $\vec{\boldsymbol{\epsilon}}_R^{\mathsf{f}} = \{\epsilon_{Rj}^{\mathsf{f}}\}_{j=1}^{M}$, and $\vec{\boldsymbol{\epsilon}}_O^{\mathsf{f}} = \{\epsilon_{Oj}^{\mathsf{f}}\}_{j=1}^{M}$. We define the induced integer scaling vector:

$$\vec{\boldsymbol{\theta}} \coloneqq \left\lfloor 2^{2k} \cdot (\vec{\boldsymbol{\epsilon}}_L^{\mathsf{f}} \circ \vec{\boldsymbol{\epsilon}}_R^{\mathsf{f}}/\vec{\boldsymbol{\epsilon}}_O^{\mathsf{f}}) \right\rfloor \in \mathbb{F}^M.$$

The vector $\vec{\boldsymbol{\theta}}$ is treated as a committed NN parameter. For each operation type t, we select the type-t scalings as $\boldsymbol{b}^{(t)}\vec{\boldsymbol{\theta}}$. We obtain quantized HD-R1CS by adding a scaling–rounding layer to each HD-R1CS equation in Eqs. (11)–(13), *i.e.*, we left-multiply by $\mathsf{Diag}_n((\boldsymbol{b}^{(t)}\vec{\boldsymbol{\theta}}) \otimes \mathbf{1}_n)$ and apply $\mathsf{Clamp}_k(\mathsf{Trunc}_{2k}(\cdot))$ element-wise.

For instance, quantized HD-R1CS for matrix multiplication is:

$$\begin{aligned}\mathsf{Diag}_n([(\boldsymbol{b}^{\mathsf{mul}}\boldsymbol{a}_O) \otimes \boldsymbol{I}_n]\boldsymbol{w}) = \mathsf{Clamp}_k(\mathsf{Trunc}_{2k}(\mathsf{Diag}_n((\boldsymbol{b}^{\mathsf{mul}}\vec{\boldsymbol{\theta}}) \otimes \mathbf{1}_n) \\ \cdot \mathsf{Diag}_n([(\boldsymbol{b}^{\mathsf{mul}}\boldsymbol{a}_L) \otimes \boldsymbol{I}_n]\boldsymbol{w}) \cdot \mathsf{Diag}_n([(\boldsymbol{b}^{\mathsf{mul}}\boldsymbol{a}_R) \otimes \boldsymbol{I}_n]\boldsymbol{w}))\end{aligned}$$

Analogous quantized HD-R1CS constraints are formulated for other operations except for the nonlinear activation function.

For a nonlinear activation function $\phi(\cdot)$, denote the scaling factors of its input and output by $\epsilon_{\mathsf{in}}^{\phi}$ and $\epsilon_{\mathsf{out}}^{\phi}$, respectively. Define the induced integer-valued function:

$$\phi'(x) \coloneqq \lfloor(1/\epsilon_{\mathsf{out}}^{\phi}) \cdot \phi(\epsilon_{\mathsf{in}}^{\phi} \cdot x)\rfloor.$$

Then, quantized HD-R1CS for ϕ in a quantized NN is written as:

$$[(\boldsymbol{b}^{\phi}\boldsymbol{a}_O) \otimes \boldsymbol{I}_n]\boldsymbol{w} = \phi'([(\boldsymbol{b}^{\phi}\boldsymbol{a}_L) \otimes \boldsymbol{I}_n]\boldsymbol{w}),$$

which is proven via sparse lookup proofs (Appendix C.4).

8 Experiments

We implement zkSMART in Rust and evaluate it on a cloud server with an Intel Xeon 6982P-C CPU (64 threads) and 128 GiB RAM. For baselines that exceed 128 GiB RAM, we use a second cloud server with the same CPU configuration and 512 GiB RAM. Unless stated otherwise, provers use 64 threads and verifiers are single-threaded.

Benchmark family. We encode HD-R1CS constraints for a quantized multilayer perceptron (MLP) with depth D. Each layer computes $\boldsymbol{X}_{i+1} = \phi(\boldsymbol{W}_i\boldsymbol{X}_i + \boldsymbol{\beta}_i)$, where ϕ is the sigmoid activation. We use the 8-bit quantization primitives from Section 7. We prove activations via the sparse lookup proof in Appendix C.4, *i.e.*, via the induced integer-valued function ϕ' from Section 7.

We use a reference encoder for this model family rather than a general ONNX-to-HD-R1CS compiler. This isolates proof-system overhead from frontend engineering and model-specific compilation choices. All layers are homogeneously sized in our implementation. For each configuration (D, N), we choose a square width n (a power of two) and set each layer to use an $n \times n$ weight matrix and an n-dimensional bias vector, so that the total parameter count satisfies $N \approx D \cdot (n^2 + n)$. In the depth sweep in Figure 5(i), we fix N and adjust n as a function of D. In the parameter sweep in Figure 5(ii), we fix D and vary n to realize the target N.

PCS and representation. We use a Dory variant [25] over BLS12-381 for memory efficiency. In our implementation, committing to 10^9 8-bit integers uses 3.46 GB of RAM, whereas other PCSs in arkworks require at least 102 GB in the tested configurations. This gap comes from the representation path in the PCS backend. Our Dory-based backend commits to packed 8-bit values in the evaluated path, while the tested arkworks PCS backends materialize an expanded vector of field elements, inflating peak RAM.

We vary depth D from 2^2 to 2^{14} and total parameter count N from 2^{20} to 2^{32}. We report setup, NN commitment (structure and parameters), prover time, proof size, verifier time, and peak RAM. Unless stated otherwise, "prover time" refers to online proving and includes witness commitment, proof reduction (PIOP), and PCS openings. Table 3 further reports architecture commitment and architecture opening as separate line items in the cost breakdown. In Table 4, we exclude architecture opening to align with baselines that do not open an architecture commitment.

Comparison with an existing architecture-hiding zkML system. Among architecture-hiding zkML systems, Evalyn reports experiments on ∼1B-parameter MLPs, while pR1CS reports smaller VGG16 networks (about 140M parameters). We therefore compare zkSMART with Evalyn [24], which hides architectures via a proof-of-proof (PoP) strategy. We use an MLP with $D = 2^{10}$ layers and width $n = 2^{10}$ (about 1B parameters). Table 3 summarizes the results.

On this benchmark, zkSMART achieves prover time 926.50 s, proof size 367.50 KB, and single-thread verifier time 153.31 ms. Architecture commitment and opening take 9.26 s and 18.48 s, respectively, and are small relative to the end-to-end prover time at this scale.

Relative to Evalyn, zkSMART reduces architecture commitment and opening time on this benchmark. zkSMART also has lower prover time in Table 3, consistent with avoiding PoP overhead

Table 3: Performance on a 2^{10}-layer MLP (∼1B parameters)

Layer i: $\boldsymbol{X}_i \leftarrow \phi(\boldsymbol{W}_i\boldsymbol{X}_{i-1} + \boldsymbol{\beta}_i)$	Evalyn	zkSMART
Unverified NN inference	3.55 s	3.56 s
Setup time	2.42 s	4.85 s
NN commitment time	2,023.87 s	1,425.68 s
Architecture commitment	69.32 s	9.26 s
Parameter commitment	1,281.85 s	1,416.42 s
NN commitment size	42.27 KB	6.34 KB
Prover time	1,412.25 s	926.50 s
Witness commitment	731.86 s	213.72 s
Proof reduction	218.80 s	563.66 s
# Fiat–Shamir challenges	*25,609*	*1,829*
PCS opening	25.37 s	119.58 s
Architecture opening	(PoP) 97.36 s	18.48 s
# Fiat–Shamir proving	*332.71 s*	N/A
Proof size	96.52 KB	367.50 KB
Verifier time (single thread)	173.15 ms	153.31 ms
Peak RAM	82.74 GB	104.93 GB

Parallelized across 64 CPU threads unless noted as "single thread"

Table 4: Comparison with non-architecture-hiding zkML systems on a 2^{10}-layer MLP (∼1B parameters)

	Prover Time	Proof Size	Verifier Time	Peak RAM
zkCNN [47][b]	1933.44 s	9.63 MB	0.68 s	626.21 GB
MSCProof [5][c]	280.51 s	2.05 MB	5.95 s	96.88 GB
zkSMART	897.37 s	0.35 MB	0.16 s	102.17 GB

[a] Provers use 64-thread CPUs; verifiers are single-threaded
[b] zkCNN out of memory at 2^{10} layers; extrapolated from 2^9 layers
[c] MSCProof proves no activations, only linear layers

in the main proving path. Witness commitment is faster in our implementation, reflecting our Dory PCS engineering choices.

zkSMART has a larger proof size in Table 3. This difference reflects compression choices in the reported configurations. Evalyn applies PoP-style compression, while our reported zkSMART numbers do not apply an additional PoP compression layer to the zkSMART transcript. Table 3 also reports that zkSMART uses 14× fewer Fiat–Shamir challenges than Evalyn. Accordingly, adding PoP-style compression to zkSMART is a plausible way to reduce proof size, but we do not rely on this for our main claims.

Figure 5 evaluates scalability under depth and parameter sweeps. In our current implementation, zkSMART scales favorably with depth D at fixed N (Figure 5(i)). Proof size and verifier time scale less favorably than Evalyn in the reported configurations, largely because Evalyn applies PoP-style compression while zkSMART does not. Accordingly, Figure 5 compares scalability under these configurations, without claiming that the gap vanishes.

Comparison with non-architecture-hiding zkML systems. To examine the overhead of architecture hiding, we compare zkSMART

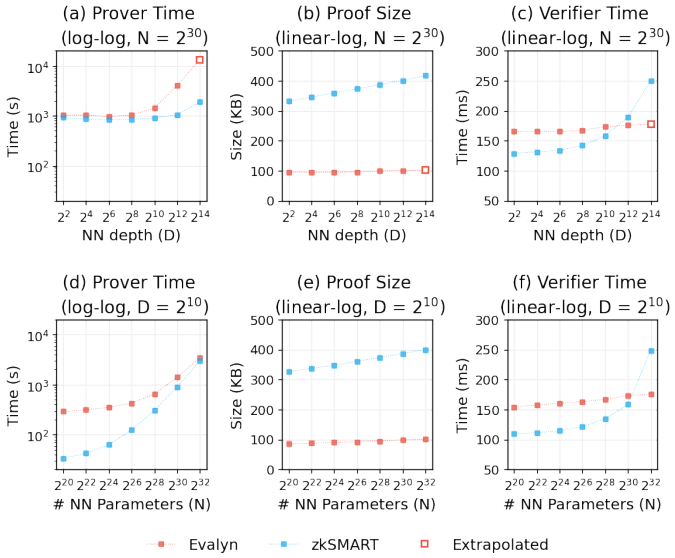


Figure 5: Comparison of Evalyn and zkSMART under (i) varying depth D with fixed total parameter count N, and (ii) varying N with fixed depth D

with two non-architecture-hiding zkML systems, zkCNN [47] and MSCProof [5], in Table 4. These systems both support fully connected layers and can be evaluated on our MLP benchmark.

On the same setting, zkCNN and MSCProof take 1933.44,s and 280.51,s, respectively. zkCNN runs out of memory at $D = 2^{10}$ layers, so Table 4 extrapolates from $D = 2^9$ by linear scaling in depth (doubling time and peak RAM when doubling D at fixed width). In Table 4, MSCProof is 3.2× faster than zkSMART, but it does not support nonlinear activations.

Without an architecture commitment, proof size and verifier work need not be independent of the (hidden) architecture description. Table 4 reflects this on our benchmark. Compared to zkSMART, systems without an architecture commitment require at least 5.8× larger proofs (in bytes) and at least 4.2× slower single-thread verification time (in wall-clock time) in the reported configuration.

9 Conclusion

We introduced HD-R1CS, a matrix-oriented constraint system designed for matrix-intensive computations such as NN inference. To commit to an NN with M matrix operations, HD-R1CS requires only $M \times M$ sparse matrices with $O(M)$ nonzero entries, thereby avoiding the substantial overhead incurred by large sparse matrices in scalar-based constraint systems. As a result, HD-R1CS supports committing to and hiding NN architectures with asymptotically optimal $O(M)$ prover time and memory usage.

We further developed zkSMART, a zkSNARK protocol for HD-R1CS. For matrix circuits consisting of M matrices of size $n \times n$, zkSMART achieves $O(n^2M)$ prover time and $O(\log(nM))$ proof size and verifier time. To improve practical efficiency, zkSMART incorporates several structure-aware proof techniques tailored to HD-R1CS.

Finally, we extended HD-R1CS to quantized HD-R1CS, integrating industry-standard quantized NNs. This enables succinct zkML proofs for quantized NN inference within the HD-R1CS framework.

Acknowledgments

The first two authors acknowledge ChatGPT for language editing.

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. 2024. Zero-Knowledge Proofs of Training for Deep Neural Networks. In *CCS*. 4316–4330.
[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *CCS*. 2087–2104.
[3] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. 2024. STIR: Reed-Solomon Proximity Testing with Fewer Queries. In *CRYPTO*. 380–413.
[4] Thomas Attema, Ronald Cramer, and Lisa Kohl. 2021. A Compressed Σ-Protocol Theory for Lattices. In *Crypto Part II*. 549–579.
[5] David Balbás, Dario Fiore, María Isabel González Vasco, Damien Robissout, and Claudio Soriente. 2023. Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing. In *CCS*. 1437–1451.
[6] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP*. 14:1–14:17.
[7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable Zero Knowledge with No Trusted Setup. In *Crypto Part III*. 701–732.
[8] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2019. Aurora: Transparent Succinct Arguments for R1CS. In *Eurocrypt Part I*. 103–128.
[9] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *TCC (B2)*. 31–60.
[10] Eli Ben-Sasson and Madhu Sudan. 2008. Short PCPs with Polylog Query Complexity. *SIAM J. Comput.* 38, 2 (2008), 551–607.
[11] Manuel Blum, Will Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. 1994. Checking the correctness of memories. *Algorithmica* 12 (1994), 225–244.
[12] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In *Eurocrypt Part II*. 327–357.
[13] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. 2020. Linear-Time Arguments with Sublinear Verification from Tensor Codes. In *TCC Part II*. 19–46.
[14] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. 2022. Gemini: Elastic SNARKs for Diverse Environments. In *Eurocrypt Part II*. 427–457.
[15] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. 2022. Zero-Knowledge IOPs with Linear-Time Prover and Polylogarithmic-Time Verifier. In *Eurocrypt Part II*. 275–304.
[16] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. 2025. Blaze: Fast SNARKs from Interleaved RAA Codes. In *EUROCRYPT*. 123–152.
[17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *S&P*. 315–334.
[18] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK Compilers. In *Eurocrypt Part I*. 677–706.
[19] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. 2021. Proofs for Inner Pairing Products and Applications. In *Asiacrypt Part III*. 65–97.
[20] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. 2024. Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees. In *PKC*. 337–369.
[21] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *Eurocrypt Part II*. 499–530.
[22] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *Eurocrypt Part I*. 738–768.
[23] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. 2020. Fractal: Post-Quantum and Transparent Recursive Proofs from Holography. In *Eurocrypt Part I*. 769–793.
[24] Mingshu Cong, Sherman S. M. Chow, Siu-Ming Yiu, and Tsz Hon Yuen. 2025. Scalable zkSNARKs for Matrix Computation: A Generic Framework for Verifiable Deep Learning. In *Asiacrypt Part V*. 363–395.
[25] Mingshu Cong, Tsz Hon Yuen, and Siu-Ming Yiu. 2025. DualMatrix: Conquering zkSNARK for Large Matrix Multiplication. IACR Cryptol. ePrint Arch. 2025/1768.
[26] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS*. 90–112.
[27] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. ZEN: An Optimizing Compiler for Verifiable, Zero-Knowledge Neural Network Inferences. IACR Cryptol. ePrint Arch. 2021/087.
[28] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time. In *IFIP Congress*. 839–842.
[29] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-Bases for Oecumenical Noninteractive Arguments of Knowledge. IACR Cryptol. ePrint Arch. 2019/953.
[30] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. 2023. Experimenting with Zero-Knowledge Proofs of Training. In *CCS*. 1880–1894.
[31] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. 2022. Succinct Zero Knowledge for Floating Point Computations. In *CCS*. 1203–1216.
[32] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs Without PCPs. In *Eurocrypt*. 626–645.
[33] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud. In *NIPS*. 4672–4681.
[34] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. 2015. Delegating computation: Interactive proofs for muggles. *J. ACM* 62, 4 (2015), 1–64.
[35] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. 2021. Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS. In *Crypto Part II*. 193–226.
[36] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR (Poster)*. 11 pages.
[37] Jens Groth. 2016. On the Size of Pairing-Based Non-Interactive Arguments. In *Eurocrypt Part II*. 305–326.
[38] Yanpei Guo, Zhanpeng Guo, Wenjie Qu, and Jiaheng Zhang. 2025. Architecture-private Zero-knowledge Proof of Neural Networks. IACR Cryptol. ePrint Arch. 2025/2212.
[39] Ulrich Haböck. 2022. Multivariate Lookups Based on Logarithmic Derivatives. IACR Cryptol. ePrint Arch. 2022/1530.
[40] Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. 2024. Scalable Zero-knowledge Proofs for Non-linear Functions in Machine Learning. In *Usenix Sec*. 3819–3836.
[41] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*. 2704–2713.
[42] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. Scaling up Trustless DNN Inference with Zero-Knowledge Proofs. arXiV 2210.08674.
[43] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Asiacrypt*. 177–194.
[44] Julien Keuffer, Refik Molva, and Hervé Chabanne. 2018. Efficient Proof Composition for Verifiable Computation. In *ESORICS*. 152–171.
[45] Jonathan Lee. 2021. Dory: Efficient, Transparent Arguments for Generalised Inner Products and Polynomial Commitments. In *TCC Part I*. 1–34.
[46] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2023. vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs. *TDSC* 21, 4 (2023), 4254–4270.
[47] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *CCS*. 2968–2985.
[48] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. 1990. Algebraic Methods for Interactive Proof Systems. In *FOCS*. 2–10.
[49] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings. In *CCS*. 2111–2128.
[50] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. 2022. FP8 formats for deep learning. arXiv 2209.05433.
[51] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *AsiaCCS*. 506–519.
[52] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly Practical Verifiable Computation. *Commun. ACM* 59, 2 (2016), 103–112.
[53] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *Crypto Part III*. 704–737.
[54] Srinath Setty and Jonathan Lee. 2020. Quarks: Quadruple-Efficient Transparent zkSNARKs. IACR Cryptol. ePrint Arch. 2020/1275.
[55] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. 2023. Customizable constraint systems for succinct arguments. IACR Cryptol. ePrint Arch. 2023/552.
[56] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. 2024. Unlocking the Lookup Singularity with Lasso. In *Eurocrypt Part VI*. 180–209.
[57] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkLLM: Zero Knowledge Proofs for Large Language Models. In *CCS*. 4405–4419.
[58] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR Poster*. 10 pages.
[59] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *Crypto Part II*. 71–89.
[60] Florian Tramèr, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. 2017. The Space of Transferable Adversarial Examples. arXiV 1704.03453. , 15 pages.

[61] Alexander Vlasov and Konstantin Panarin. 2019. Transparent Polynomial Commitment Scheme with Polylogarithmic Communication Complexity. IACR Cryptol. ePrint Arch. 2019/1020.
[62] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. 2013. A Hybrid Architecture for Interactive Verifiable Computation. In *S&P*. 223–237.
[63] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. 2017. Full Accounting for Verifiable Outsourcing. In *CCS*. 2071–2086.
[64] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *S&P*. 926–943.
[65] Suppakit Waiwitlikhit, Ion Stoica, Yi Sun, Tatsunori Hashimoto, and Daniel Kang. 2024. Trustless Audits without Revealing Data or Models. In *ICML*. 49808–49821.
[66] Haodi Wang and Thang Hoang. 2023. ezDPS: An Efficient and Zero-Knowledge Machine Learning Inference Pipeline. *Proc. Priv. Enhancing Technol.* 2023, 2 (2023), 430–448.
[67] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *Usenix Sec*. 501–518.
[68] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. 2023. pvCNN: Privacy-Preserving and Verifiable Convolutional Neural Network Testing. *TIFS* 18 (2023), 2218–2233.
[69] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Crypto Part II*. 733–764.
[70] Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2022. Orion: Zero Knowledge Proof with Linear Prover Time. In *Crypto Part IV*. 299–328.
[71] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8Bit BERT. In *NeurIPS*. 36–39.
[72] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup Arguments in Sublinear Time. In *CCS*. 3121–3134.
[73] Hadas Zeilberger, Binyi Chen, and Ben Fisch. 2024. BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes. In *CRYPTO*. 138–169.
[74] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2020. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *S&P*. 859–876.
[75] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *S&P*. 863–880.

Open Science

Artifact overview. This appendix enumerates the artifacts needed to evaluate the core contributions of HD-R1CS and zkSMART and to reproduce the performance results reported in the paper.

Artifacts.

- Source code implementing HD-R1CS and the zkSMART proving and verification algorithms.
- A reference encoder that generates HD-R1CS instances for the evaluated MLP benchmark family.
- Benchmark drivers to run the prover and verifier and to sweep depth D and total parameter count N as in Section 8.
- Measurement and logging code that reports setup time, commitment time, prover time, verifier time, proof size, and peak RAM, consistent with Tables 3–4 and Figure 5.
- Archived logs and CSV summaries used to generate Tables 3–4 and Figure 5, plus a plotting script to regenerate the figure from the CSV files.
- Documentation describing how to build the code, run the benchmarks, and interpret the outputs.

Access during double-blind review. The artifacts are available at https://anonymous.4open.science/r/hdr1cs-0311. No credentials are required to access the repository. This repository is a frozen snapshot intended for reproducing the paper's experiments.

Reproducibility instructions. Reviewers should follow the build and run instructions in the repository documentation. The intended reproduction targets are Tables 3–4 and the scalability trends in Figure 5. Reproducing the largest MLP configuration requires sufficient RAM to match the reported peak usage and enough CPU parallelism to match the reported thread count.

Snapshot. We built and ran the code on Ubuntu 22.04 with Rust `nightly-2025-04-28`. We built in release mode via `cargo build –workspace –release`.

To reproduce the ∼1B-parameter MLP run in Table 3, set the depth to `1024` and the weight shape to `1024x1024` in `hdr1cs/example/experiment.rs`. Run:

```
RAYON_NUM_THREADS=64
cargo run -p hdr1cs –example experiment
–release
```

The program prints setup, commitment, proving, verification, proof size, and peak-RSS measurements to stdout.

To reproduce the depth sweep in Figure 5(i) and the parameter sweep in Figure 5(ii), either rerun the above command at each sweep point by updating `depth` and `shape` in `experiment.rs` under `hdr1cs/example/`, or use the archived logs and CSV summaries in `hdr1cs/example/` and `python/`. To regenerate Figure 5 from the CSV summaries, install Python 3.10+ with `matplotlib`, `seaborn`, and `pandas`, then run `python/plot.py`, which writes `python/experiment.png`.

What is not shared and why. We do not provide a general ONNX-to-HD-R1CS compiler, since the paper treats a full frontend as outside scope and evaluates a reference encoder for the studied model family. We do not redistribute third-party baseline implementations, since they are external artifacts with their own licenses and distribution channels. The paper's comparisons can be evaluated by running zkSMART from the provided repository and comparing against baseline results reported by the respective systems.

Ethics, safety, and privacy. The artifacts contain no user data and require no private datasets. The evaluation focuses on proof-system cost for quantized NN inference rather than on model accuracy under quantization.

A Related Work

Circuit satisfiability. The circuit satisfiability (Circ-SAT) problem asks whether a circuit has an accepting assignment that satisfies all constraints. In zkSNARKs, computations are arithmetized into an arithmetic relation (an intermediate representation, IR) that is then proven and verified. Common IRs include quadratic arithmetic programs (QAP) [32], rank-1 constraint systems (R1CS) [12], the PLONK arithmetization [29], algebraic intermediate representation [7, 10], and customizable constraint systems [55]. The arithmetic circuit size scales with the running time of the underlying computation. Custom gates [21] pack multiple arithmetic gates into one constraint for a richer function. However, custom gates do not remove the intrinsic $O(n^2)$ scalar output size of an $n \times n$ matrix

operation. Thus, even with custom gates, an $n \times n$ matrix operation still requires $O(n^2)$ constraints to represent its n^2 scalar outputs.

General-purpose zkSNARK. General-purpose zkSNARKs target Circ-SAT, and their prover time scales with the arithmetic circuit size. Early-generation zkSNARKs [2, 8, 18, 22, 23, 29, 32, 37, 49, 52, 74] use quasilinear field operations, largely due to fast Fourier transforms (FFTs). Recent FFT-free systems, including Libra [69], Quarks [54], Gemini [14], Spartan [53], Brakedown [35], and HyperPlonk [21], achieve strictly linear prover time in the circuit size. Many modern zkSNARKs follow a polynomial interactive oracle proof (PIOP) [9, 18] plus polynomial commitment scheme (PCS) [43] paradigm. These PIOPs often build on the sumcheck protocol [48], which was later adopted in the zkSNARK literature [26].

Polynomial commitment scheme (PCS). PCSs include pairing-based schemes rooted in KZG [43] and folding-based constructions such as Bulletproofs [17], BMMTV21 [19], and Dory [45]. Hyrax [64] reduces prover-side cryptographic work for multilinear evaluation, while Sona [56] shifts work into additional proofs to reduce Hyrax's verification. FRI and code-based techniques [6, 61] yield post-quantum PCSs such as BCG20 [13], BCL22 [15], Brakedown [35], Orion [70], STIR [3], BaseFold [73], and Blaze [16]. These often have fast commitment and opening, but larger proof sizes and verifier times than ECC-based PCSs, and significant RAM overhead.

In this paper, we implement a specific variant [25] of Dory [45]. It is memory efficient and delivers prover performance comparable to code-based PCSs for short-bitwidth integer matrices, while keeping logarithmic proof size and verifier time. This choice balances prover time, proof size, verifier time, and RAM usage in our setting. Nevertheless, the HD-R1CS framework is PCS-agnostic and is compatible with any PCS that offers succinct proofs.

Zero-knowledge machine learning (zkML). zkML proves correctness of neural network (NN) inference or training in zero knowledge. Early zkML solutions [33, 67] were not succinct. Achieving succinctness typically increases prover time. A standard approach converts NNs into arithmetic circuits, as in Zen [27], but the resulting circuits can be impractically large for large NNs. Recent zkML systems [1, 5, 30, 42, 44, 46, 47, 57, 68] instead prove smaller NN components to reduce prover cost. They optimize matrix multiplication, convolution, and nonlinear activations to further reduce prover time. Floating-point computations can be converted to integer computations via range proofs [31].

B Background

B.1 Definitions

Definition 6 (Argument of Knowledge). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is called an argument of knowledge for the relation $\mathcal{R}$ if it satisfies perfect completeness and computational knowledge soundness.*

Definition 7 (Perfect Completeness). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has perfect completeness if for any non-uniform PPT adversary $\mathcal{A}$:*

$$\Pr\left(\begin{array}{l|l} (\mathsf{srs}, u, w) \notin \mathcal{R} \quad \vee & \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda); \\ \mathcal{V}(\mathsf{srs}, u; \pi) = \mathsf{true} & (u, w) \leftarrow \mathcal{A}(\mathsf{srs}); \\ & \pi \leftarrow (\mathcal{P} \rightleftharpoons \mathcal{V})(\mathsf{srs}, u; w) \end{array}\right) = 1,$$

where srs, u, w, *and* π *stand for the structured reference string, the statement, the witness, and the transcript, respectively.*

Definition 8 (Knowledge Soundness [4]). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *for relation $\mathcal{R}$ is knowledge sound with error $\kappa_\lambda(|u|) \colon \mathbb{N} \to [0,1]$ if there exists an algorithm $\mathcal{K}$, called a knowledge extractor, given a statement u and black-box oracle access to a (potentially dishonest) prover $\mathcal{P}^*$, it runs in an expected number of steps polynomial in $|u|$ (counting queries to $\mathcal{P}$ as a single step) and outputs a witness $(\mathsf{srs}, u; w) \in \mathcal{R}$ with probability:*

$$\Pr((\mathsf{srs}, u \colon \mathcal{K}(u)) \in \mathcal{R}) \geq \frac{p(\mathcal{P}^*, u) - \kappa_\lambda(|u|)}{\mathsf{poly}(|u|)},$$

where $p(\mathcal{P}^*, u)$ is the probability that $\mathcal{P}^*$ generates a valid proof.

Definition 9 (Public-Coin). *A $(2\mu+1)-$move public-coin interactive protocol $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is public-coin if in the $(2j)$-th move, $\mathcal{V}$ sends the j-th challenge $x_j \xleftarrow{\$} \mathbb{X}_j$. All challenges $x_1, \ldots, x_\mu$ sent from the verifier to the prover are public information and chosen uniformly at random and independently of the prover's messages.*

Definition 10 (Adaptive Fiat–Shamir Transformation). *The adaptive Fiat–Shamir transformation on a $(2\mu+1)-$move public-coin interactive protocol: $\mathsf{FS}(\mathsf{Setup}, \mathcal{P}, \mathcal{V}) \mapsto (\mathsf{Setup}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ is a two-round argument of knowledge where $\mathcal{P}_{\mathsf{FS}}(\mathsf{srs}, u, w)$ runs $\mathcal{P}(\mathsf{srs}, u, w)$ but instead of asking the verifier for the challenge x_i for message a_i, the challenges are computed as:*

$$x_j = \mathsf{Hash}_i(u, a_1, \ldots, a_{j-1}), \quad j \in [\mu].$$

the output is then the transcript $\pi = (a_1, \ldots, a_{\mu-1}, a_\mu)$. On input of a statement u and a transcript $\pi = (a_1, \ldots, a_{\mu-1}, a_\mu)$, $\mathcal{V}_{\mathsf{FS}}(\mathsf{srs}, u, \pi)$ accepts if, for $x_j, j \in [\mu]$ as above, $\mathcal{V}$ accepts the transcript $\pi = (a_1, \ldots, a_{\mu-1}, a_\mu)$ on input u.

Definition 11 (Perfect Zero-Knowledge). *A non-interactive argument of knowledge $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is perfect zero-knowledge [37] if there exists a PPT simulator $\mathcal{S}$ and a trapdoor td such that for all stateful distinguishers $\mathcal{D}$ the following two probabilities are equal:*

$$\begin{aligned} &\Pr\left(\mathcal{D}(\mathsf{srs}, u; \pi; \mathsf{td}) = \mathsf{true} \;\middle|\; \begin{array}{l} (\mathsf{srs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow (\mathcal{P} \rightleftharpoons \mathcal{V})(\mathsf{srs}, u; w) \end{array}\right) \\ =&\Pr\left(\mathcal{D}(\mathsf{srs}, u; \pi^*; \mathsf{td}) = \mathsf{true} \;\middle|\; \begin{array}{l} (\mathsf{srs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi^* \leftarrow \mathcal{S}(\mathsf{srs}, u; \mathsf{td}) \end{array}\right). \end{aligned}$$

Definition 12 (zkSNARK). *A zero-knowledge non-interactive argument of knowledge $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ is a succinct non-interactive argument of knowledge (zkSNARK) if:*

- *The setup time is quasilinear in witness size;*
- *The prover time is quasilinear in witness size;*
- *The proof size is polylogarithmic in witness size;*
- *The verifier time is polylogarithmic in witness size.*

B.2 Sumcheck Protocol

The sumcheck protocol operates on a degree-d, μ-variate polynomial $\psi(\boldsymbol{x}) \in \mathbb{F}[X^\mu]$ and verifies that the sum of $\psi(\boldsymbol{x})$ over the

Algorithm 5: Sumcheck Protocol

Input: Public input: $\sigma \in \mathbb{F}, d \in \mathbb{N}$;
$\mathcal{P}$'s private input: $\psi \in \mathbb{F}[X^\mu]$
Output: true or false

$\sigma_{(0)} \leftarrow \sigma \in \mathbb{F}$;
foreach $j \in [1, \mu]$ **do**
  $\mathcal{P}$ computes a degree-d univariate polynomial:
    $\psi^{(j)}(x) \leftarrow \sum_{(x_{j+1},\ldots,x_\mu)\in\{0,1\}^{\mu-j}} \psi(x_{(1)},\ldots,x_{(j-1)},x,x_{j+1},\ldots,x_\mu)$,
    where $x_{(1)},\ldots,x_{(j-1)}$ denote challenges from prior $(j-1)$ iterations;
  $\mathcal{P} \to \mathcal{V}$: $\psi^{(j)}(x) \in \mathbb{F}[X]$;
  $\mathcal{V}$: **if** $\sigma_{(j-1)} \neq \psi^{(j)}(0) + \psi^{(j)}(1)$ **then return** false;
  $\mathcal{V} \to \mathcal{P}$: $x_{(j)} \xleftarrow{\$} \mathbb{F}^*$;
  $\mathcal{P}$ and $\mathcal{V}$ compute: $\sigma_{(j)} \leftarrow \psi^{(j)}(x_{(j)}) \in \mathbb{F}$;
$\mathcal{P}$ produces additional proofs to let $\mathcal{V}$ verify that $\sigma_{(\mu)}$ is the evaluation of ψ at $(x_{(1)}, x_{(2)}, \ldots, x_{(\mu)})$;

Boolean hypercube $\boldsymbol{x} \in \{0,1\}^\mu$ equals a given value $\sigma \in \mathbb{F}$. The sumcheck relation $\mathcal{R}_{\mathsf{sumcheck}}$ is:

$$\left\{ \left(\sigma \in \mathbb{F}\right) : \left(\psi \in \mathbb{F}[X^\mu]\right) \middle| \left(\sigma = \sum_{\boldsymbol{x}\in\{0,1\}^\mu} \psi(\boldsymbol{x})\right) \right\}.$$

The protocol in Algorithm 5 reduces the sumcheck relation to checking a polynomial evaluation $\psi(\boldsymbol{x})$ at a randomly chosen point $\boldsymbol{x}$, which is then proven via the Open algorithm of a PCS.

Existing research demonstrates that sumcheck protocols on MLEs can effectively prove the inner product, Hadamard product, and matrix multiplication. Below, we summarize the findings:

B.2.1 Frobenius Inner Product. For the Frobenius inner product $c = \boldsymbol{a} \bullet \boldsymbol{b}$, where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^{n\times n}$, we have:

$$\{c = \boldsymbol{a} \bullet \boldsymbol{b}\} \iff \left\{ c = \sum_{\boldsymbol{x}\in\{0,1\}^{2\log_2 n}} \widehat{\boldsymbol{a}}(\boldsymbol{x}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{x}) \right\}.$$

The inner product proof is designed as follows:

Inner product proof for $\boldsymbol{c} = \boldsymbol{a} \bullet \boldsymbol{b}$
- $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol that reduces $c = \sum_{\boldsymbol{x}\in\{0,1\}^{2\log_2 n}} \widehat{\boldsymbol{a}}(\boldsymbol{x}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{x})$ into checking $v = \widehat{\boldsymbol{a}}(\boldsymbol{x}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{x})$ at the challenge point $\boldsymbol{x}$;
- $\mathcal{P} \to \mathcal{V}$: $v_a \leftarrow \widehat{\boldsymbol{a}}(\boldsymbol{x})$ and $v_b \leftarrow \widehat{\boldsymbol{b}}(\boldsymbol{x})$; $\mathcal{V}$ checks $v = v_a \cdot v_b$;
- $\mathcal{P}$ and $\mathcal{V}$ run protocols to verify $v_a = \widehat{\boldsymbol{a}}(\boldsymbol{x})$ and $v_b = \widehat{\boldsymbol{b}}(\boldsymbol{x})$;

B.2.2 Hadamard Product. For the Hadamard product $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b} \in \mathbb{F}^{n\times n}$, using the Schwartz–Zippel lemma and random tensor-structured vector $\vec{\boldsymbol{l}}, \vec{\boldsymbol{r}} \in \mathbb{F}^n$ derived from $\boldsymbol{x}_l, \boldsymbol{x}_r \in \mathbb{F}^{\log_2 n}$ as per Eq. (6), we check:

$$\{\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}\} \iff \left\{ \vec{\boldsymbol{l}}^{\mathsf{T}} \boldsymbol{c} \vec{\boldsymbol{r}} = (\boldsymbol{a} \circ \boldsymbol{b}) \bullet (\vec{\boldsymbol{l}} \otimes \vec{\boldsymbol{r}}^{\mathsf{T}}) \right\} \iff$$

$$\left\{ \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \sum_{\boldsymbol{y},\boldsymbol{z}\in\{0,1\}^{\log_2 n}} \widehat{\boldsymbol{a}}(\boldsymbol{y},\boldsymbol{z}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{z}) \cdot \widehat{\vec{\boldsymbol{l}}}(\boldsymbol{y}) \cdot \widehat{\vec{\boldsymbol{r}}}(\boldsymbol{z}) \right\},$$

noting that $\widehat{\vec{\boldsymbol{l}}}(\boldsymbol{y})$ and $\widehat{\vec{\boldsymbol{r}}}(\boldsymbol{z})$ correspond to the equality polynomial in Spartan [53].

With this equivalence, the Hadamard product proof is as follows:

Hadamard product proof for $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}$
- $\mathcal{V} \to \mathcal{P}$: $\boldsymbol{x}_l, \boldsymbol{x}_r \in \mathbb{F}^{\log_2 n}$; $\mathcal{P} \to \mathcal{V}$: $v_c \leftarrow \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) \in \mathbb{F}$;
- $\mathcal{P}$ computes $\vec{\boldsymbol{l}}$ and $\vec{\boldsymbol{r}}$ in Eq. (6);
- $\mathcal{P}$ and $\mathcal{V}$ run a sumcheck protocol that reduces $v_c = \sum_{\boldsymbol{y},\boldsymbol{z}\in\{0,1\}^{\log_2 n}} \widehat{\boldsymbol{a}}(\boldsymbol{y},\boldsymbol{z}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{z}) \cdot \widehat{\vec{\boldsymbol{l}}}(\boldsymbol{y}) \cdot \widehat{\vec{\boldsymbol{r}}}(\boldsymbol{z})$ into checking $v_a = \widehat{\boldsymbol{a}}(\boldsymbol{y},\boldsymbol{z})$ and $v_b = \widehat{\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{z})$ at points $\boldsymbol{y}$ and $\boldsymbol{z}$;
- $\mathcal{P}$ and $\mathcal{V}$ run protocols to verify $v_a = \widehat{\boldsymbol{a}}(\boldsymbol{y},\boldsymbol{z})$ and $v_b = \widehat{\boldsymbol{b}}(\boldsymbol{y},\boldsymbol{z})$;

B.2.3 Matrix Multiplication. For an $n \times n$ matrix multiplication $\boldsymbol{c} = \boldsymbol{ab}$, by employing random tensor-structured vectors $\vec{\boldsymbol{l}} \in \mathbb{F}^n$ and $\vec{\boldsymbol{r}} \in \mathbb{F}^n$ derived from the $\boldsymbol{x}_l \in \mathbb{F}^{\log_2 n}$ and $\boldsymbol{x}_r \in \mathbb{F}^{\log_2 n}$ in Lemma 1, we have:

$$\{\boldsymbol{c} = \boldsymbol{ab}\} \iff \left\{ \vec{\boldsymbol{l}}^{\mathsf{T}} \boldsymbol{c} \vec{\boldsymbol{r}} = \vec{\boldsymbol{l}}^{\mathsf{T}} \boldsymbol{ab} \vec{\boldsymbol{r}} \right\} \iff$$

$$\left\{ \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \sum_{\boldsymbol{x}\in\{0,1\}^{\log_2 n}} \widehat{\boldsymbol{a}}(\boldsymbol{x}_l, \boldsymbol{x}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{x}, \boldsymbol{x}_r) \right\}.$$

The matrix multiplication proof was proposed by Thaler [59] and is detailed in Section 5.3 of the main text.

B.2.4 Addition and Transposition. The arguments for addition and transposition are trivial to derive through MLEs:

$$\{\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}\} \iff \left\{ \widehat{\boldsymbol{c}}(\boldsymbol{x}) = \widehat{\boldsymbol{a}}(\boldsymbol{x}) + \widehat{\boldsymbol{b}}(\boldsymbol{x}) \right\}.$$

$$\left\{ \boldsymbol{c} = \boldsymbol{a}^{\mathsf{T}} \right\} \iff \left\{ \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \widehat{\boldsymbol{a}}(\boldsymbol{x}_r, \boldsymbol{x}_l) \right\}.$$

Matrix addition proof for $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$
- $\mathcal{V} \to \mathcal{P}$: $\boldsymbol{x} \in \mathbb{F}^{2\log_2 n}$;
- $\mathcal{P} \to \mathcal{V}$: $v_c \leftarrow \widehat{\boldsymbol{c}}(\boldsymbol{x}), v_a \leftarrow \widehat{\boldsymbol{a}}(\boldsymbol{x}), v_b \leftarrow \widehat{\boldsymbol{b}}(\boldsymbol{x}) \in \mathbb{F}$;
- $\mathcal{V}$: **if** $v_c \neq v_a + v_b$ **then return** false;
- $\mathcal{P}$ and $\mathcal{V}$ run protocols to verify $v_a = \widehat{\boldsymbol{a}}(\boldsymbol{x})$ and $v_b = \widehat{\boldsymbol{b}}(\boldsymbol{x})$;

Matrix transposition proof for $\boldsymbol{c} = \boldsymbol{a}^{\mathsf{T}}$
- $\mathcal{V} \to \mathcal{P}$: $\boldsymbol{x}_l, \boldsymbol{x}_r \in \mathbb{F}^{\log_2 n}$;
- $\mathcal{P} \to \mathcal{V}$: $v \leftarrow \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) \in \mathbb{F}$;
- $\mathcal{P}$ and $\mathcal{V}$ run protocols to verify $v = \widehat{\boldsymbol{a}}(\boldsymbol{x}_r, \boldsymbol{x}_l)$;

B.2.5 Convolution. With zero-padding of matrices, the convolution of matrices $\boldsymbol{a} \in \mathbb{F}^{n_a\times n_a}$ and $\boldsymbol{b} \in \mathbb{F}^{n_b\times n_b}$ is defined by:

$$\mathsf{Conv}(\boldsymbol{a}, \boldsymbol{b}) \coloneqq \left(\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} a_{ij} b_{(I-i),(J-j)} \right)_{I=1,J=1}^{n_a+n_b-1,n_a+n_b-1}.$$

Utilizing the Schwartz–Zippel lemma, convolution is proven via $n \geq n_a + n_b$ and random challenges $y, y' \in \mathbb{F}^*$:

$$\vec{\boldsymbol{y}}^{\mathsf{T}}_{[:n_a+n_b]} \boldsymbol{c} \vec{\boldsymbol{y}'}_{[:n_a+n_b]} = \vec{\boldsymbol{y}}^{\mathsf{T}}_{[:n_a+n_b]} \mathsf{Conv}(\boldsymbol{a}, \boldsymbol{b}) \vec{\boldsymbol{y}'}_{[:n_a+n_b]},$$

where: $\vec{y} = (1, y, y^2, \ldots, y^{n-1}), \vec{y'} = (1, y', y'^2, \ldots, y'^{n-1})$.

Straightforward mathematical derivation yields:

$$\begin{aligned}
&\vec{y}^{\top}_{[:n_a+n_b]}\mathsf{Conv}(\boldsymbol{a},\boldsymbol{b})\vec{y'}_{[:n_a+n_b]}\\
=&\sum_{I=-\infty}^{\infty}\sum_{J=-\infty}^{\infty}\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} a_{ij}b_{(i-I),(j-J)}y^{-I}y'^{-J}\\
=&\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} a_{ij}y^{-i}y'^{-j}\sum_{I=-\infty}^{\infty}\sum_{J=-\infty}^{\infty} b_{(i-I),(j-J)}y^{-(i-I)}y'^{-(j-J)}\\
=&\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} a_{ij}y^{-i}y'^{-j}\sum_{I=-\infty}^{\infty}\sum_{J=-\infty}^{\infty} b_{IJ}y^{-I}y'^{-J}\\
=&(\vec{y}^{\top}_{[:n_a]}\boldsymbol{a}\vec{y'}_{[:n_a]})\cdot(\vec{y}^{\top}_{[:n_b]}\boldsymbol{b}\vec{y'}_{[:n_b]})
\end{aligned}$$

Consequently, the convolution proof is established through:

$$\begin{aligned}
&\{\boldsymbol{c} = \mathsf{Conv}(\boldsymbol{a},\boldsymbol{b})\} \iff\\
&\left\{\vec{y}^{\top}_{[:n_a+n_b]}\boldsymbol{c}\vec{y'}_{[:n_a+n_b]} = (\vec{y}^{\top}_{[:n_a]}\boldsymbol{a}\vec{y'}_{[:n_a]})\cdot(\vec{l}^{\top}_{[:n_b]}\boldsymbol{b}\vec{y'}_{[:n_b]}), \forall y, y'\right\}.
\end{aligned}$$

Padding the matrices with zeros to form $n \times n$ matrices, this equation becomes:

$$\vec{y}^{\top}\boldsymbol{c}\vec{y'} = (\vec{y}^{\top}\boldsymbol{a}\vec{y'})\cdot(\vec{l}^{\top}\boldsymbol{b}\vec{y'}).$$

Here, $\vec{y}^{\top}\boldsymbol{c}\vec{y'}$, $\vec{y}^{\top}\boldsymbol{a}\vec{y'}$, and $\vec{l}^{\top}\boldsymbol{b}\vec{y'}$ are MLE evaluations at the point

$$(1, y^2, y^4, \ldots, y^{2^{\log n}-1}, 1, y'^2, y'^4, \ldots, y'^{2^{\log n}-1}).$$

Convolution proof for $\boldsymbol{c} = \mathsf{Conv}(\boldsymbol{a},\boldsymbol{b})$

- $\mathcal{V} \to \mathcal{P}$: $y, y' \in \mathbb{F}^*$; $\mathcal{P}$ and $\mathcal{V}$ compute $\vec{y}, \vec{y'}$;
- $\mathcal{P} \to \mathcal{V}$: $v_c \leftarrow \vec{y}^{\top}\boldsymbol{c}\vec{y'}, v_a \leftarrow \vec{y}^{\top}\boldsymbol{a}\vec{y'}, v_b \leftarrow \vec{y}^{\top}\boldsymbol{b}\vec{y'} \in \mathbb{F}$;
- $\mathcal{V}$: **if** $v_c \neq v_a \cdot v_b$ **then return** false;
- $\mathcal{P}$ runs the Open algorithm of a PCS to let $\mathcal{V}$ verify $v_c = \vec{y}^{\top}\boldsymbol{c}\vec{y'}$, $v_a = \vec{y}^{\top}\boldsymbol{a}\vec{y'}$, and $v_b = \vec{y}^{\top}\boldsymbol{b}\vec{y'}$;

For M convolutions $\boldsymbol{c}_i = \mathsf{Conv}(\boldsymbol{a}_i, \boldsymbol{b}_i), i \in [M]$, they are collectively verified by checking:

$$\begin{pmatrix} \vec{y}^{\top}\boldsymbol{c}_1\vec{y'} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \vec{y}^{\top}\boldsymbol{c}_M\vec{y'}\end{pmatrix} = \begin{pmatrix} \vec{y}^{\top}\boldsymbol{a}_1\vec{y'} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \vec{y}^{\top}\boldsymbol{a}_M\vec{y'}\end{pmatrix} \circ \begin{pmatrix} \vec{y}^{\top}\boldsymbol{b}_1\vec{y'} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \vec{y}^{\top}\boldsymbol{b}_M\vec{y'}\end{pmatrix}.$$

Here:

$$\begin{aligned}
\begin{pmatrix} \vec{y}^{\top}\boldsymbol{c}_1\vec{y'} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \vec{y}^{\top}\boldsymbol{c}_M\vec{y'}\end{pmatrix} &= \begin{pmatrix} \vec{y}^{\top} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \vec{y}^{\top}\end{pmatrix}\begin{pmatrix} \boldsymbol{c}_1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \boldsymbol{c}_M\end{pmatrix}\begin{pmatrix} \vec{y'} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \vec{y'}\end{pmatrix}\\
&= [\boldsymbol{I}_M \otimes \vec{y}^{\top}]\begin{pmatrix} \boldsymbol{c}_1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \boldsymbol{c}_M\end{pmatrix}[\boldsymbol{I}_M \otimes \vec{y'}],
\end{aligned}$$

Similar equations can be formulated for the right-hand side of the above batch convolution check.

B.3 Existing Sparse Arithmetization

Marlin [22] and Spartan [53] use the following arithmetization to reduce a sparse bilinear form to a dense inner product:

$$\widehat{\boldsymbol{a}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \vec{\boldsymbol{l}}^{\top}\boldsymbol{a}\vec{\boldsymbol{r}} = \vec{\boldsymbol{a}} \bullet (\vec{\boldsymbol{l}}_a \circ \vec{\boldsymbol{r}}_a).$$

Here, $\vec{\boldsymbol{a}} \in \mathbb{F}^N$ lists the N non-zero entries of $\boldsymbol{a} \in \mathbb{F}^{n\times n}$. If $\vec{\boldsymbol{a}}[\tau] = \boldsymbol{a}[i, j]$, then $\vec{\boldsymbol{l}}_a[\tau] = \vec{\boldsymbol{l}}[i]$ and $\vec{\boldsymbol{r}}_a[\tau] = \vec{\boldsymbol{r}}[j]$.

In Spartan, lookup proofs enforce that $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$ are consistent with $\vec{\boldsymbol{l}}$ and $\vec{\boldsymbol{r}}$ at the specified indices. Concretely, given commitments to $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$, the prover proves each entry is selected from $\vec{\boldsymbol{l}}$ or $\vec{\boldsymbol{r}}$ at a predetermined position. Because $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$ have full bitwidth, committing to them is costly for large sparse matrices.

The correct constructions of $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$ are ensured by lookup proofs in Spartan. Given commitments to $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$, lookup proofs verify that entries of $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$ are retrieved from $\vec{\boldsymbol{l}}$ and $\vec{\boldsymbol{r}}$ at prespecified positions. Since $\vec{\boldsymbol{l}}_a$ and $\vec{\boldsymbol{r}}_a$ are full-bitwidth vectors, committing to them incurs a significant overhead for large sparse matrices.

C Lookup Proof

C.1 Error-Proving by Range Proofs

Error-proving was employed to construct zero-knowledge proofs for floating-point computations [31]. The core idea applies range proofs to the rounding errors of floating-point truncation.

The Trunc_k in Eq. (19) is proved by a matrix range proof. Let $\boldsymbol{a} \in \mathbb{F}^{(Mn)\times n}$ be a dense field matrix, and let $\boldsymbol{c} \in \mathbb{F}^{(Mn)\times n}$ represent the matrix truncated by the last k bits. Then, proving $\boldsymbol{c} = \mathsf{Trunc}_k(\boldsymbol{a})$ is equivalent to proving that the error matrix $\boldsymbol{e}$, as defined below, is at most k-bit:

$$\boldsymbol{e} = \boldsymbol{a} - 2^k \cdot \boldsymbol{c}, \quad |\boldsymbol{e}| < 2^k \cdot \boldsymbol{1}.$$

Similarly, the Clamp_k function in Eq. (19) is proved by matrix range proofs. To prove $\boldsymbol{c} = \mathsf{Clamp}_k(\boldsymbol{a})$, the prover commits to the signs of $\boldsymbol{a}$, denoted by $\boldsymbol{\sigma} \in \{-1, 1\}^{(Mn)\times n}$, and proves the following equations and inequalities:

$$\begin{aligned}
&\boldsymbol{0} \le \boldsymbol{\sigma} \circ \boldsymbol{c} < 2^{k-1} \cdot \boldsymbol{1},\ \boldsymbol{\sigma} \circ (\boldsymbol{a} - \boldsymbol{c}) \ge \boldsymbol{0},\\
&(\boldsymbol{a} - \boldsymbol{c}) \circ [\boldsymbol{\sigma} \circ \boldsymbol{c} - (2^{k-1} - 1) \cdot \boldsymbol{1}] = \boldsymbol{0}.
\end{aligned}$$

Besides the range proofs, proofs for the Hadamard products (Appendix B.2) are also required.

As noted by Lasso [56], range proofs can be derived from lookup proofs. For instance, the range proof for $|\boldsymbol{e}| < 2^k \cdot \boldsymbol{1}$ is equivalent to proving that all entries of $\boldsymbol{e}$ are looked up from the set $\{-2^k + 1, \ldots, 2^k - 1\}$. We therefore discuss lookup proofs.

C.2 Lookup Proof of Lasso

Nonlinear NN activations are typically handled via lookup proofs [40, 57]. Spartan [53] uses lookup proofs in its sparse-matrix arithmetization for R1CS. Most lookup proofs either use memory checking [11, 35, 40, 53, 56] or encode set inclusion as arithmetic constraints [20, 39, 57, 72]. Lasso [56] is a recent memory-checking based lookup proof that builds on Spartan [53]. Below, we restate Lasso's lookup argument as matrix equations.

A lookup proof lets a prover commit to $\vec{c} \in \mathbb{F}^m$ and prove that each entry lies in a fixed table $\vec{v} \in \mathbb{F}^n$. To express this relation, define $\vec{s} \in \mathbb{F}^n$ where s_i counts occurrences of v_i in $\vec{c}$. Define $\vec{t} \in \mathbb{F}^m$ where t_i counts occurrences of c_i in the prefix $\{c_1, \ldots, c_{i-1}\}$. By construction, $0 \le s_i \le m$ for all $i \in [n]$, and $0 \le t_i \le m$ for all $i \in [m]$. We therefore commit to $\vec{s}$ and $\vec{t}$ using a bounded bitwidth. Verification reduces to checking that two value-pair vectors are permutations of each other:

$$((\vec{c} \,\|\, \vec{v}), (\vec{t} \,\|\, \vec{s})), \quad ((\vec{v} \,\|\, \vec{c}), (\vec{0} \,\|\, (\vec{t} + \vec{1}))).$$

This check uses a random challenge $z \in \mathbb{F}^*$, ensuring the grand products of entries from the following vectors are identical:

$$\vec{1} + z(\vec{c} \| \vec{v}) + z^2(\vec{t} \| \vec{s}), \quad \vec{1} + z(\vec{v} \| \vec{c}) + z^2(\vec{0} \| (\vec{t} + \vec{1})).$$

We use a grand product proof for:

$$\mathcal{L}_{\text{gp}} = \left\{ c \in \mathbb{F}, \vec{a} \in \mathbb{F}^n \mid c = \Pi_{i=1}^n a_i, \right\}$$

which is proven by committing to $\vec{w} \in \mathbb{F}^{n-1}$ and proving:

$$\begin{pmatrix} \vec{w} \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ \vec{w} \end{pmatrix} \circ \boldsymbol{a}.$$

C.3 Sparse Lookup Proof

A sparse lookup proof attests that all nonzero entries of a vector $\vec{c} \in \mathbb{F}^m$ appear in a lookup table $\vec{v} \in \mathbb{F}^n$. Let $\vec{\eta} \in \{0,1\}^m$ be the indicator vector of nonzero positions in $\vec{c}$, *i.e.*, $\eta_i = 1$ iff $c_i \neq 0$.

The sparse lookup is verified in two steps. First, sparsity is enforced by checking:

$$\vec{c} \circ (\vec{1} - \vec{\eta}) = \vec{0},$$

which ensures that $\vec{c}$ is zero outside the indicated positions.

Second, we verify that every nonzero entry of $\vec{c}$ appears in $\vec{v}$ using a grand-product argument. Specifically, we check equality of the grand products of the following pairs:

$$\vec{1} + z(\vec{c} \| \vec{v}) + z^2(\vec{t} \| \vec{s}), \quad \vec{1} + z(\vec{v} \| \vec{c}) + z^2(\vec{0} \| (\vec{t} + \vec{\eta})).$$

Here $\vec{t}$ is defined analogously to Lasso, except that $t_i = 0$ whenever $c_i = 0$, exploiting sparsity to reduce commitment overhead.

C.4 Lookup Proof for Nonlinear Function

As noted in prior work such as zkLLM [40, 57], nonlinear activation functions can be handled with lookup proofs. Let $(\vec{k}, \vec{v})$ denote the lookup table containing all key-value pairs of the activation function ϕ. Proving $\vec{c} = \phi(\vec{a})$ then reduces to showing that each pair (a_i, c_i) appears in the table $(\vec{k}, \vec{v})$.

To support sparse inputs, we modify the lookup table to store key-value pairs of the shifted function $\phi(x + \phi^{-1}(0))$, ensuring that the pair $(0, 0)$ is included in the table. Under this construction, proving $\vec{c} = \phi(\vec{a})$ is equivalent to proving that

$$(\vec{a} - \phi^{-1}(0) \cdot \vec{1},\ \vec{c})$$

appears in the lookup table.

D Proofs of Propositions

D.1 Proof of Proposition 1

Proof. Let $(\vec{l}, \vec{r})$ be computed from $(\boldsymbol{x}_l, \boldsymbol{x}_r)$ as per Eq. (6) of Lemma 1, $(\vec{l}_a, \vec{r}_a)$ be computed from $(\boldsymbol{x}_{al}, \boldsymbol{x}_{ar})$ and $(\vec{l}_b, \vec{r}_b)$ from $(\boldsymbol{x}_{bl}, \boldsymbol{x}_{br})$. Then, we know:

$$\vec{l} = \vec{l}_a \otimes \vec{l}_b, \quad \vec{r} = \vec{r}_a \otimes \vec{r}_b.$$

Leveraging Eq. (5), we have:

$$\begin{aligned} \widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) =& (\vec{l}_a \otimes \vec{l}_b)^\top (\boldsymbol{a} \otimes \boldsymbol{b})(\vec{r}_a \otimes \vec{r}_b) \\ =& (\vec{l}_a{}^\top \boldsymbol{a} \vec{r}_a) \cdot (\vec{l}_b{}^\top \boldsymbol{b} \vec{r}_b) = \widehat{\boldsymbol{a}}(\boldsymbol{x}_{al}, \boldsymbol{x}_{ar}) \cdot \widehat{\boldsymbol{b}}(\boldsymbol{x}_{bl}, \boldsymbol{x}_{br}). \end{aligned}$$

Here, we utilize the mixed-product property between Kronecker product and matrix multiplication. □

D.2 Proof of Proposition 2

Proof. As per Eq. (5) of Lemma 5, for arbitrary ($\boldsymbol{x}_l \in \mathbb{F}^{\log_2(nM)}$, $\boldsymbol{x}_r \in \mathbb{F}^{\log_2(nM)}$) and tensor-structured ($\vec{l} \in \mathbb{F}^{nM}, \vec{r} \in \mathbb{F}^{nM}$) derived from $(\boldsymbol{x}_l, \boldsymbol{x}_r)$ as per Eq. (6), we have:

$$\widehat{\boldsymbol{c}}(\boldsymbol{x}_l, \boldsymbol{x}_r) = \vec{l}^\top \boldsymbol{c} \vec{r}.$$

The tensor structure of $\vec{l}$ and $\vec{r}$ allows decomposition:

$$\vec{l} = \vec{l}_a \otimes \vec{l}_b, \quad \vec{r} = \vec{r}_a \otimes \vec{r}_b, \quad \vec{l}_a, \vec{r}_a \in \mathbb{F}^M, \quad \vec{l}_b, \vec{r}_b \in \mathbb{F}^n,$$

where $(\vec{l}_a, \vec{r}_a)$ and $(\vec{l}_b, \vec{r}_b)$ are respectively computed from $(\boldsymbol{x}_{al}, \boldsymbol{x}_{ar})$ and $(\boldsymbol{x}_{bl}, \boldsymbol{x}_{br})$ according to Eq. (6).

Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_M$ be the M blocks of $\boldsymbol{a}$. Then:

$$\begin{aligned} \vec{l}^\top \boldsymbol{c} \vec{r} &= \begin{pmatrix} l_{a1}\vec{l}_b{}^\top & \cdots & l_{aM}\vec{l}_b{}^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{a}_M \end{pmatrix} \begin{pmatrix} r_{a1}\vec{r}_b \\ \vdots \\ r_{aM}\vec{r}_b \end{pmatrix} \\ &= \begin{pmatrix} l_{a1}r_{a1}\vec{l}_b{}^\top & \cdots & l_{aM}r_{aM}\vec{l}_b{}^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_M \end{pmatrix} \vec{r}_b \\ &= [(\vec{l}_a \circ \vec{r}_a) \otimes \vec{l}_b]^\top \boldsymbol{a} \vec{r}_b. \end{aligned}$$

Then, direct arithmetic shows that by setting $\boldsymbol{x}'_l$ as in Eq. (16):

$$\begin{aligned} \vec{l}_a \circ \vec{r}_a = [&\prod_{j=1}^{\log_2 M} \frac{(1 - x_{alj})(1 - x_{arj})}{1 - x'_{lj}}] \\ &\cdot [(1 - x'_{l1}, x'_{l1}) \otimes \cdots \otimes (1 - x'_{l\log_2 M}, x'_{l\log_2 M})]. \end{aligned}$$

Utilizing Eq. (5), we derive Eq. (15). □