Zero-Knowledge AI Inference with High Precision

Arman Riasi
Virginia Tech
Blacksburg, VA, USA
armanriasi@vt.edu

Haodi Wang
City University of Hong Kong
Kowloon Tong, Hong Kong
haodi.wang@cityu.edu.hk

Rouzbeh Behnia
University of South Florida
Tampa, FL, USA
behnia@usf.edu

Viet Vo
Swinburne University of Technology
Melbourne, Australia
vvo@swin.edu.au

Thang Hoang
Virginia Tech
Blacksburg, VA, USA
thanghoang@vt.edu

Abstract

Artificial Intelligence as a Service (AIaaS) enables users to query a model hosted by a service provider and receive inference results from a pre-trained model. Although AIaaS makes artificial intelligence more accessible, particularly for resource-limited users, it also raises verifiability and privacy concerns for the client and server, respectively. While zero-knowledge proof techniques can address these concerns simultaneously, they incur high proving costs due to the non-linear operations involved in AI inference and suffer from precision loss because they rely on fixed-point representations to model real numbers.

In this work, we present ZIP, an efficient and precise commit and prove zero-knowledge SNARK for AIaaS inference (both linear and non-linear layers) that natively supports IEEE-754 double-precision floating-point semantics while addressing reliability and privacy challenges inherent in AIaaS. At its core, ZIP introduces a novel relative-error-driven technique that efficiently proves the correctness of complex non-linear layers in AI inference computations without any loss of precision, and hardens existing lookup-table and range proofs with novel arithmetic constraints to defend against malicious provers. We implement ZIP and evaluate it on standard datasets (e.g., MNIST, UTKFace, and SST-2). Our experimental results show, for non-linear activation functions, ZIP reduces circuit size by up to three orders of magnitude while maintaining the full precision required by modern AI workloads.

CCS Concepts

• **Security and privacy**; • **Computing methodologies** → **Machine learning**; **Artificial intelligence**;

Keywords

Zero-Knowledge Proofs, Numerical Methods, Non-linear Function

Final Version. This is the final version of the paper accepted at ACM CCS 2025. The published version is available at https://doi.org/10.1145/3719027.3765056.

1 Introduction

Artificial Intelligence (AI) has demonstrated exceptional performance across a wide range of domains, including data analysis and pattern recognition. However, training state-of-the-art machine learning models requires high-quality data, domain expertise, and specialized computing infrastructure, which many organizations may lack. To overcome these challenges, many turn to Artificial Intelligence as a Service (AIaaS) platforms. AIaaS enables the commercialization of artificial intelligence capabilities through scalable cloud resources, allowing organizations to benefit from advanced capabilities such as analytics, automation, and predictive insights without relying on in-house expertise or infrastructure.

While AIaaS alleviates the burden of maintaining local infrastructure, it introduces a critical trust assumption: users have no direct way to verify that their inputs are processed correctly by the intended model. As a result, they must implicitly trust the provider to deliver correct and faithful inference results. However, this assumption is strong and can be violated in practice. An adversarial provider can reduce processing costs by substituting a lightweight model or fabricating outputs entirely. Even well-intentioned providers can constitute a single point of failure. For example, insider threats, software misconfigurations, insecure deployments (e.g., vulnerable AI libraries), or malware may allow attackers to inject code, alter model parameters, or tamper with outputs, thereby compromising the integrity of the entire AI inference pipeline [62]. These concerns are especially critical in high-stakes domains like medical diagnosis, financial decisions, or fraud detection, where one tampered output can lead to severe consequences. While publishing the model would enable verification, doing so exposes proprietary intellectual property, such as model weights, which service providers cannot afford due to commercial sensitivity. Therefore, there is a critical need for a verifiable and secure inference mechanism that ensures the correct execution of the user's input on the server's intended model, while preserving model confidentiality.

To address the verifiability challenges, existing methods (e.g., [29, 76]) often leverage Verifiable Computation (VC) [13, 63], which enables the server to provide a mathematical proof that the inference computation was performed correctly on the intended AI model. Although VC enables clients to verify the inference integrity, the proof partially exposes the server's model, risking intellectual-property leakage. Zero-knowledge machine learning (zkML) [23, 37, 39, 74] addresses this leakage issue by employing zero-knowledge proof (ZKP) techniques [7, 9, 25, 28, 46] to preserve model privacy while still proving correct inference computation.

Despite their strong integrity and privacy guarantees, existing ZKP methods may not be suitable for handling non-linear operations, potentially becoming a bottleneck in AI inference proofs when compared to linear computations. Specifically, existing ZKP constructions for non-linear functions require large circuits and suffer from precision loss since they rely on converting floating-point models to fixed-point encodings over finite fields to represent model real numbers. When sensitive applications such as healthcare or finance are considered, even a tiny precision loss, whether from fixed-point representation or approximations of complex non-linear functions, can have serious ramifications for organizations [4, 47, 61]. For instance, in medical diagnostics, a model might distinguish benign from malignant lesions based on subtle pixel differences [4, 61].

Existing studies have attempted to address these bottlenecks for relatively simple non-linear activations (e.g., ReLU, sigmoid, tanh), but suffer from inherent drawbacks. One line of research uses linear built-ins (e.g., gadgets) or bit decomposition [23, 37, 39, 66] to encode non-linear functions, which becomes prohibitively expensive as activations grow more complex. Another approach employs piecewise-polynomial approximations [2, 26, 29, 76], reducing circuit size but introducing approximation errors that degrade precision when used alone. A third strategy relies on precomputed lookup tables [12, 30, 41, 57] or multiple client-server interactions [51] for activation computations, both of which incur large storage overhead, scalability issues, or high communication costs.

Meanwhile, AI research is increasingly adopting more complex non-linear activations such as ELU, SeLU, and GeLU. As a result, state-of-the-art verifiable secure inference frameworks [2, 12, 30, 37, 39, 41, 51, 57, 66, 76] find themselves trapped in a "trade-off triangle" of scalability, efficiency, and precision to support modern activation functions without exploding proving time, or sacrificing model accuracy. Given these limitations in proving complex non-linear functions within AI inference, we pose the following question:

Can we design a zero-knowledge AI inference scheme that efficiently proves complex non-linear function computations without sacrificing accuracy or functionality, while guaranteeing both inference integrity and server model privacy?

1.1 Our Contributions

In this paper, we answer the above question affirmatively by presenting ZIP, a new framework for privacy-preserving and verifiable AI inference. ZIP supports the verification of full AI inference pipelines on complex models that are evaluated under high numerical precision, while simultaneously preserving model privacy. In particular, ZIP offers the following key properties:

- **IEEE-754-compliant AI Inference Pipeline:** A key property of ZIP is that it permits zero-knowledge verification of AI inference evaluated under IEEE-754 double-precision semantics. This allows ZIP to offer higher precision performance (e.g., accuracy), at the cost of longer proving/verification time, than prior techniques [30, 39, 41, 57, 66] that trade precision using fixed-point representations for faster proving/verifying AI inference, which suffer from quantization errors.
- **Reduced Constraints for Non-Linear Activations:** ZIP only requires a few thousand R1CS constraints to prove non-linear activations in zero-knowledge under IEEE-754 semantics. This achieves up to three orders of magnitude reduction in circuit size compared to prior gadget-based or bit-decomposition approaches when directly applied to prove the same non-linear activations under IEEE-754 standards [23, 37, 39, 66].
- **Activation-Agnostic Support:** ZIP supports a wide range of modern activation functions that are critical in AI inference pipelines, such as GeLU, SeLU, and ELU. Another important aspect of our proposed technique is that it can generalize to other activations without requiring custom circuit redesign. This offers greater flexibility than prior approaches that require designing new gadget [23, 37, 39, 66] for each non-linear function.
- **Small Lookup Table Size:** ZIP only requires a small lookup table to enable ZKP of non-linear activations under IEEE-754 double-precision. For example, to prove GeLU, ZIP requires 70 table entries. This is one to three orders of magnitude smaller than prior schemes [30, 41, 57] for proving non-linear activations, which require hundreds to thousands of lookup table entries.

Technical Highlights. To enable efficient zero-knowledge proofs of AI inference computed under high-precision IEEE-754 semantics, we come up with a new technique based on numerical analysis and relative error bounds. The idea is as follows. Given a non-linear activation function, we approximate it using piecewise polynomials and store their coefficients in a lookup table. During inference, the server locates the polynomial piece corresponding to a given input and evaluates it to produce an approximate output. However, to preserve numerical fidelity throughout the AI inference pipeline, the server does not use this approximation for downstream computation. Instead, it uses the exact computation produced by the full-precision IEEE-754 evaluation of the activation function. The server then proves (in zero-knowledge) that the approximate output is within a small relative error of the exact computation. This design decouples the proof cost from the complexity of the activation function while preserving high computational precision.

In essence, our design requires proving two key statements: *(i)* lookup validity, which shows that the selected polynomial is present in the predefined lookup table and corresponds to the input interval; and *(ii)* approximation soundness, which shows that the approximated value is very close to the exact computation.

While there exist some techniques that can partially support lookup validity, applying them in our AI inference setting with model privacy guarantees is non-trivial. Specifically, the server must prove that it has selected a vector of coefficients corresponding to a single polynomial piece, without revealing which piece or interval was chosen. Although standard lookup arguments (e.g., [71]) allow proving that multiple entries exist in a table, they do not enforce ordering among them. This limitation becomes critical when polynomial coefficients are stored as individual entries in the lookup table, as an adversary could combine coefficients from different polynomial pieces and produce a valid proof. On the other hand, naively applying a standard range proof with a hidden interval may allow an adversary to mix endpoints from different intervals in the table, resulting in a valid proof for an invalid interval that does not correspond to any actual polynomial piece.

To address the ordered selection challenge, our approach is to store each coefficient of every polynomial piece as a separate entry in the lookup table. The entries are organized such that: (i) coefficients within a polynomial piece appear in fixed order, and (ii) all the pieces themselves follow the order of their corresponding input intervals. This layout preserves both the internal structure of each polynomial and the global order of the piecewise polynomial. We then design new arithmetic constraints on top of existing lookup arguments to ensure that the selected entries belong to the same polynomial piece and are retrieved in the correct order (see §4.3.1).

To address the hidden interval challenge, we represent the interval endpoints of polynomial pieces in a separate public lookup table. Each pair of endpoints is stored in two consecutive entries, and these pairs are ordered according to the order of their corresponding polynomial pieces. We then design new arithmetic constraints on top of the existing lookup argument to ensure that the selected private endpoints are adjacent in the table, retrieved in order, and that the secret input lies within the selected interval (see §4.3.2).

To prove approximation soundness, we adopt the relative error model from [27] and apply Horner's method [32] to efficiently prove polynomial evaluations under IEEE-754 semantics (see §4.3.3).

Finally, we note that a full AI inference pipeline consists of both linear and non-linear layers. Our contribution primarily targets non-linear activation functions due to their complex structure and the difficulty of proving them efficiently. For linear layers, ZIP directly leverages the existing technique from [20] to prove IEEE-754-compliant linear operations.

We formalize the security of our technique and implement and evaluate ZIP on real datasets. The source code and experimental scripts are available at https://github.com/vt-asaplab/ZIP.

2 Preliminaries

Notation. Let $\mathbb{Z}_n = \{0, 1, 2, \cdots, n\}$, $\mathbb{Z}_n^* = \mathbb{Z}_n \setminus \{0\}$, and $\mathbb{N} = \{0, 1, 2, \cdots\}$. λ denotes security parameters, and $\mathsf{negl}(\cdot)$ stands for the negligible function. $x \overset{\$}{\leftarrow} \mathbb{Z}_n$ indicates that x is chosen randomly from the set $\mathbb{Z}_n$. PPT refers to Probabilistic Polynomial Time. $A \overset{c}{\approx} B$ indicates computational indistinguishability of two quantities A and B. Bold letters (like $\mathbf{a}$ and $\mathbf{A}$) denote vector and matrix, respectively. $\mathbf{a}[i]$ represents the element i in vector $\mathbf{a}$. We define a polynomial f^k of degree k with coefficients $\{a_i\}_{i=0}^k$ by $f^k(x) = \sum_{i=0}^k a_i x^i$.

2.1 Commit-and-Prove Argument Systems

Commit-and-Prove SNARK (CP-SNARK) [9] permits a prover to commit to a witness w in advance for an NP-statement $(\mathcal{R}, \mathcal{L})$, where $\mathcal{R}$ is an NP relation and $\mathcal{L}$ is the corresponding language of valid inputs. Given an input $x \in \mathcal{L}$, the prover proves the knowledge of a witness w such that $(x, w) \in \mathcal{R}$. CP-SNARK is a tuple of PPT algorithms CPZKP = (Setup, Comm, Prov, Ver) as follows.

- $\mathsf{pp} \leftarrow \mathsf{CPZKP.Setup}(1^\lambda)$: Given a security parameter λ, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{CPZKP.Comm}(\mathbf{w}, r, \mathsf{pp})$: Given a witness $\mathbf{w}$ and randomness r, it outputs cm as a commitment to $\mathbf{w}$.
- $\pi \leftarrow \mathsf{CPZKP.Prov}(\mathbf{x}, \mathbf{w}, \mathsf{pp})$: Given a statement $\mathbf{x}$ and a witness $\mathbf{w}$, it outputs a proof π that shows $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$.
- $b \leftarrow \mathsf{CPZKP.Ver}(\pi, \mathsf{cm}, \mathbf{x}, \mathsf{pp})$: Given a proof π, a commitment cm, and a statement $\mathbf{x}$, it outputs b where b is 1 (accept) if π is a valid proof for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and cm is a commitment to $\mathbf{w}$; otherwise, it outputs 0 (reject).

Security. We present CP-SNARK's security properties as follows.

Definition 1. CP-SNARK satisfies the following security properties.

- *Completeness:* For any $\lambda, r, (\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ s.t. $\mathsf{pp} \leftarrow \mathsf{CPZKP.Setup}(1^\lambda)$, $\mathsf{cm} \leftarrow \mathsf{CPZKP.Comm}(\mathbf{w}, r, \mathsf{pp})$, and $\pi \leftarrow \mathsf{CPZKP.Prov}(\mathbf{x}, \mathbf{w}, \mathsf{pp})$, it holds that $\Pr[\mathsf{CPZKP.Ver}(\pi, \mathsf{cm}, \mathbf{x}, \mathsf{pp}) = 1] = 1$.
- *Knowledge soundness:* For any PPT prover $\mathcal{P}^*$, and statement $\mathbf{x}$, there exists a PPT extractor $\mathcal{E}$ that, with access to the entire execution process and the randomness of $\mathcal{P}^*$, can extract a witness $\mathbf{w}$ s.t. $\mathsf{pp} \leftarrow \mathsf{CPZKP.Setup}(1^\lambda), \mathsf{cm} \leftarrow \mathsf{CPZKP.Comm}(\mathbf{w}, r, \mathsf{pp})$ for randomness r, $\pi^* \leftarrow \mathcal{P}^*(\mathbf{x}, \mathsf{pp}), \mathbf{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathbf{x}, \pi^*, \mathsf{pp})$, it holds
$$\Pr\left[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge \mathsf{CPZKP.Ver}(\pi^*, \mathsf{cm}, \mathbf{x}, \mathsf{pp}) = 1\right] \leq \mathsf{negl}(\lambda)$$
- *Zero-knowledge:* There exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for any PPT verifier $\mathcal{V}^*$, $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, and $\mathsf{cm} \leftarrow \mathsf{CPZKP.Comm}(\mathbf{w}, r, \mathsf{pp})$, it holds that

$$\Pr\begin{bmatrix} \mathsf{pp} \leftarrow \mathsf{CPZKP.Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{CPZKP.Prov}(\mathbf{x}, \mathbf{w}, \mathsf{pp}) \\ \mathsf{CPZKP.Ver}(\pi, \mathsf{cm}, \mathbf{x}, \mathsf{pp}) = 1 \end{bmatrix} \overset{c}{\approx} \Pr\begin{bmatrix} \mathsf{pp} \leftarrow \mathcal{S}_1(1^\lambda) \\ \pi \leftarrow \mathcal{S}_2(\mathbf{x}, \mathsf{pp}) \\ \mathcal{V}^*(\pi, \mathsf{cm}, \mathbf{x}, \mathsf{pp}) = 1 \end{bmatrix}$$

2.1.1 Polynomial Commitment Scheme. Polynomial Commitment (PC) lets a prover commit to a polynomial so the prover can later prove an evaluation of the committed polynomial at a given point.

PC is commonly used in CP-SNARKs, where the prover commits to polynomial(s) that encode the witness and circuit structure. These commitments hide the polynomial coefficients but support evaluation proofs. During proof generation, the prover derives a new polynomial that aggregates the relevant circuit constraints using the values encoded in the previously committed polynomial(s), and proves that this derived polynomial evaluates to a claimed value at a randomly chosen point. The verifier, using public setup parameters, performs a single check that verifies both the correctness of the evaluation and its consistency with the original commitment.

KZG Polynomial Commitment [35]. Let $\mathbb{G}$ be a cyclic group of prime order q with generator $[\![1]\!]$, and denote any group element as $[\![x]\!] \in \mathbb{G}$ where $x \in \mathbb{Z}_q$ is its discrete logarithm relative to $[\![1]\!]$. We write group operations additively ($[\![x]\!] + [\![y]\!] = [\![x + y]\!]$) and scalar multiplication as $\delta \cdot [\![x]\!] = [\![\delta x]\!]$ for $\delta \in \mathbb{Z}_q$. We recall KZG polynomial commitment [35] that offers efficient proof and verification. This efficiency stems from its one-time trusted setup, which generates a structured reference string (SRS) for polynomials of degree up to k. The KZG scheme relies on a bilinear pairing group. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order q. Let $[\![1]\!]_1$ and $[\![1]\!]_2$ be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear pairing is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and use $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, [\![1]\!]_1, [\![1]\!]_2) \leftarrow \mathsf{BilGen}(1^\lambda)$ to output the parameters for the bilinear pairing. KZG contains the following algorithms.

- $\mathsf{pp}' \leftarrow \mathsf{Setup}(1^\lambda, k)$: Given security parameter λ and polynomial degree bound k, it runs $\mathsf{bp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, [\![1]\!]_1, [\![1]\!]_2) \leftarrow \mathsf{BilGen}(1^\lambda)$. Let $\rho \overset{\$}{\leftarrow} \mathbb{Z}_p^*$, it outputs public parameters $\mathsf{pp}' \leftarrow (\mathsf{bp}, \{[\![\rho^i]\!]_1, [\![\rho^i]\!]_2\}_{i=0}^k)$.
- $\mathsf{cm}_f \leftarrow \mathsf{Comm}(f, \mathsf{pp}')$: Given a polynomial $f(x) = \sum_{i=0}^k a_i x^i$, it outputs $\mathsf{cm}_f = [\![f(\rho)]\!]_1 = \sum_{i=0}^k a_i \cdot [\![\rho^i]\!]_1$.
- $(y, \pi_{\mathsf{KZG}}) \leftarrow \mathsf{Open}(f, \zeta, \mathsf{pp}')$: Given evaluation point ζ, it computes $q^{k-1}(x) = \frac{f(x) - f(\zeta)}{x - \zeta}$. Let $\{b_i\}_{i=0}^k$ be coefficients of q^{k-1}. It outputs $y = q^{k-1}(\zeta)$ and $\pi = [\![q^{k-1}(\rho)]\!]_1 = \sum_{i=0}^k b_i \cdot [\![\rho^i]\!]_1$.

- $\{0,1\} \leftarrow \mathsf{Ver}(\pi, \mathsf{cm}_f, \zeta, y, \mathsf{pp}')$: Given a proof π, a commitment cm, the evaluation point ζ, and the evaluation y, it outputs 1 (accept) if $e(\mathsf{cm}_f - [\![y]\!]_1, [\![1]\!]_2) = e(\pi, [\![x-\zeta]\!]_2)$; otherwise, it outputs 0 (reject).

The above KZG scheme achieves computationally hiding under the Discrete Logarithm assumption, where the commitment of the polynomial $f(x) = \sum_{i=0}^{k} a_i x^i$ is of form $\mathsf{cm} = [\![f(\rho)]\!]_1 = \sum_{i=0}^{k} a_i \cdot [\![\rho^i]\!]_1$. One can make KZG achieve unconditional hiding under the Strong Diffie-Hellman (SDH) assumption by using a random polynomial $\hat{f}(x) = \sum_{i=0}^{k} c_i x^i$ [35]. Let $[\![h]\!]_1$ be a generator in $\mathbb{G}_1$, where h remains private, the perfectly hiding commitment of $f(x)$ is $\hat{\mathsf{cm}} = [\![f(\rho)]\!]_1 + [\![\hat{f}(\rho)]\!]_1 = \sum_{i=0}^{k} a_i \cdot [\![\rho^i]\!]_1 + \sum_{i=0}^{k} c_i \cdot [\![\rho^i \cdot h]\!]_1$.

PlonK [25]. We recall PlonK, a PC-based zero-knowledge CP-SNARK framework that offers succinct, constant-time verification and streamlined proving. To enforce copy constraints (i.e., the same value appearing in multiple places in the circuit), PlonK employs permutation arguments via a zero-knowledge permutation polynomial. All constraints, including permutation checks, are aggregated into a single quotient polynomial for succinct verification.

2.1.2 *Lookup Arguments.* Lookup arguments enable a prover to prove that all elements of a committed vector $\mathbf{a} \in \mathbb{Z}_p^{k+1}$ belong to a public table $\mathbf{c} \in \mathbb{Z}_p^n$ $(k+1 < n)$; that is, for every index $\{j\}_{j=0}^{k}$, there exists some index i such that $\mathbf{a}[j] = \mathbf{c}[i]$. We recall Caulk [71], a zero-knowledge lookup argument that allows a prover to prove that each of the element of a private vector $\mathbf{a}$ belongs to a *public* table $\mathbf{c}$ without revealing either the elements of $\mathbf{a}$ or their position in $\mathbf{c}$. Caulk achieves *position-hiding linkability* between $\mathbf{a}$ and $\mathbf{c}$ by re-randomizing KZG as a vector commitment with additional blinding factors that hide both the values of $\mathbf{a}$ and their corresponding indices in $\mathbf{c}$.

2.2 Numerical Precision and Approximation

2.2.1 *Floating-Point Arithmetic.* The IEEE 754 standard [1] encodes real numbers using a finite number of bits. A floating point representation of a real number is a tuple $(\hat{s}, \hat{e}, \hat{m})$, where $\hat{s}$ is the sign bit (i.e., $\hat{s} \in \{0,1\}$), $\hat{e}$ is the exponent and $\hat{m}$ is the mantissa (or significand). In IEEE single precision, the exponent and mantissa are represented using 8 and 23 bits, respectively; in double precision, they use 11 and 52 bits, respectively.

2.2.2 *Approximation.* The relative error in numerical analysis [59] provides a normalized measure of the discrepancy between an approximate value and the exact solution. It helps determine whether an approximation is sufficiently accurate relative to the true value. Let a be the exact value and b its approximation. Given δ' as the relative error bound, the relative error is defined as $\frac{|a-b|}{|a|} \leq \delta'$.

Non-linear Functions Approximation. We recall polynomial expansions (e.g., Taylor, Chebyshev [44]) and piecewise polynomial approximations (e.g., spline methods [19]) as standard ways to approximate nonlinear functions. By tolerating a small approximation error δ, we can represent the non-linear function using either a single polynomial $f(x)$ or a set of piecewise polynomials $\mathcal{F}^m$ with m low-degree polynomials $\{f_i(x)\}_{i=1}^{m}$ over the set of intervals $\mathcal{I} = \{[t_0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]\}$, where $t_0 < t_1 < \ldots < t_m$ s.t.

$$\mathcal{F}^m(x) = \begin{cases} f_1(x), & x \in [t_0,\, t_1], \\ f_2(x), & x \in [t_1,\, t_2], \\ \vdots & \\ f_m(x), & x \in [t_{m-1},\, t_m]. \end{cases} \tag{1}$$

3 Model

System Model. Our ZIP system involves a client and a server. The server holds a well-trained AI model $\mathbf{W} \in \mathsf{M}_{\mathsf{pp}}$, where M_{pp} is the model space, and the client holds query data $\mathbf{X} \in \mathsf{N}_{\mathsf{pp}}$, with N_{pp} the message space. The server first commits to $\mathbf{W}$, after which the client sends $\mathbf{X}$ for inference. The server employs its committed model[1] to perform inference on client data while preserving the privacy of $\mathbf{W}$ and ensuring computational correctness. Formally, ZIP is a zero-knowledge AI inference scheme defined as a tuple of PPT algorithms $\mathsf{ZIP} = (\mathsf{Setup}, \mathsf{Comm}, \mathsf{Infer}, \mathsf{Ver})$ as follows.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, l)$: Given a security parameter λ and an upper bound l on model size, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{Comm}(\mathbf{W}, r, \mathsf{pp})$: Given model parameters $\mathbf{W} \in \mathsf{M}_{\mathsf{pp}}$ and randomness $r \in \mathsf{R}_{\mathsf{pp}}$, where R_{pp} is the random space, it outputs a commitment $\mathsf{cm} \in \mathsf{D}_{\mathsf{pp}}$, where D_{pp} is the commitment space.
- $(\pi, \mathrm{y}) \leftarrow \mathsf{Infer}(\mathbf{W}, \mathbf{X}, \mathsf{pp})$: Given model parameters $\mathbf{W} \in \mathsf{M}_{\mathsf{pp}}$ and query data $\mathbf{X} \in \mathsf{N}_{\mathsf{pp}}$, it outputs inference result $\mathrm{y} = \hat{\mathcal{F}}(\mathbf{W}, \mathbf{X}) \in \mathsf{N}_{\mathsf{pp}}$ (where $\hat{\mathcal{F}}$ is the ideal inference functionality) and a proof π.
- $\{0,1\} \leftarrow \mathsf{Ver}(\mathrm{y}, \mathsf{cm}, \pi, \mathsf{pp})$: Given an inference $\mathrm{y} \in \mathsf{N}_{\mathsf{pp}}$, a commitment $\mathsf{cm} \in \mathsf{D}_{\mathsf{pp}}$, and a proof π, it outputs 1 if π is a valid proof for $\mathrm{y} = \hat{\mathcal{F}}(\mathbf{W}, \mathbf{X})$; otherwise, it outputs 0.

Threat Model. In our system, there is mutual distrust between the client and the server. We assume that the server may act maliciously by processing the client's query arbitrarily, potentially using query data or model parameters other than the actual ones. On the other hand, the client is curious about the server's model parameters. We allow the client to verify the correctness of the inference results provided by the server, while ensuring the privacy of the AI model.

Security Model. We define the security model of zero-knowledge and verifiable AI inference, ensuring both the integrity of inference computation and the privacy of the server's model, as follows.

Definition 2. Zero-knowledge AI inference satisfies the following.

- **Completeness**. For any λ, l, $\mathbf{W} \in \mathsf{M}_{\mathsf{pp}}$, $\mathbf{X} \in \mathsf{N}_{\mathsf{pp}}$, $r \in \mathsf{R}_{\mathsf{pp}}$ s.t. $(\mathsf{pp}) \leftarrow \mathsf{ZIP.Setup}(1^\lambda, l)$, $\mathsf{cm} \leftarrow \mathsf{ZIP.Comm}(\mathbf{W}, r, \mathsf{pp})$, $(\pi, \mathrm{y}) \leftarrow \mathsf{ZIP.Infer}(\mathbf{W}, \mathbf{X}, \mathsf{pp})$, it holds $\Pr[\mathsf{ZIP.Ver}(\mathrm{y}, \mathsf{cm}, \pi, \mathsf{pp}) = 1] = 1$.
- **Soundness**. For any λ, l, and PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{ZIP.Setup}(1^\lambda, l) \\ (\mathsf{cm}^*, \mathbf{W}^*, \mathbf{X}, \mathrm{y}^*, \pi^*, r) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{cm}^* = \mathsf{ZIP.Comm}(\mathbf{W}^*, r, \mathsf{pp}) \\ \mathsf{ZIP.Ver}(\mathrm{y}^*, \mathsf{cm}^*, \pi^*, \mathsf{pp}) = 1 \\ \hat{\mathcal{F}}(\mathbf{W}^*, \mathbf{X}) \neq \mathrm{y}^* \end{array}\right] \leq \mathsf{negl}(\lambda)$$

[1] In practice, an external entity (e.g., an auditor) can verify the commitment in advance.

- **Model privacy**. For any λ, l, $\mathbf{W} \in \mathsf{M}_{\mathsf{pp}}$, and PPT adversary $\mathcal{A}$, there exists simulator $\mathcal{S}$ such that

$$\Pr\left[\mathcal{A}(\mathsf{y}, \mathsf{cm}, \pi, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{ZIP.Setup}(1^{\lambda}, l) \\ \mathsf{cm} \leftarrow \mathsf{ZIP.Comm}(\mathbf{W}, r, \mathsf{pp}) \\ \mathbf{X} \leftarrow \mathcal{A}(\mathsf{cm}, \mathsf{pp}) \\ (\pi, \mathsf{y}) \leftarrow \mathsf{ZIP.Infer}(\mathbf{W}, \mathbf{X}, \mathsf{pp}) \end{array}\right] \overset{c}{\approx}$$

$$\Pr\left[\mathcal{A}(\mathsf{y}, \mathsf{cm}, \pi, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{r} (\mathsf{cm}, \mathsf{pp}) \leftarrow \mathcal{S}(1^{\lambda}, l, r) \\ \mathbf{X} \leftarrow \mathcal{A}(\mathsf{cm}, \mathsf{pp}) \\ (\pi, \mathsf{y}) \leftarrow \mathcal{S}^{\mathcal{A}}(\mathsf{cm}, \mathbf{X}, r, \mathsf{pp}), \text{given} \\ \text{oracle access to } \hat{\mathcal{F}} \text{ to compute} \\ \mathsf{y} = \hat{\mathcal{F}}(\mathbf{W}, \mathbf{X}) \end{array}\right]$$

Out-of-scope attacks. In this work, we focus on ensuring the correctness of the AI inference and the privacy of the server's model parameters. Beyond this, attacks such as model extraction/stealing [10, 34, 50, 58, 64] (e.g., replicating the model) and model backdoors [52, 53] (e.g., hidden triggers) are out of scope.

4 Our Proposed ZIP Scheme

At a high level, ZIP delivers end-to-end, IEEE-754–compliant zero-knowledge proofs for AI inference on client data by introducing a novel *high-precision* technique that *efficiently* verifies the network's non-linear layers. A naïve approach to achieve high precision is to directly prove the exact computation of these non-linear functions. While conceptually simple, this approach is often impractical due to its high computational cost. An alternative is to approximate non-linear functions using numerical methods. Although this approach improves efficiency, a careless design may lead to a low inference accuracy due to precision loss caused by approximation and the limitations of the cryptographic proof back-end, which relies on finite field arithmetic (see below for details).

In this section, we show how ZIP addresses the dual challenges of efficiency and accuracy in generating zero-knowledge proofs for *non-linear* functions in AI inference.

4.1 Problem Formulation

Cryptographic operations rely on finite field arithmetic for exact computation, whereas scientific computing and machine learning operate on real numbers, typically using IEEE floating-point arithmetic. This fundamental mismatch makes fixed-point or finite field representations poorly suited for capturing the behavior of real-valued computations in AI inference. Given this, precisely proving non-linear functions (e.g., activations) in AI inference, defined over the real numbers, is challenging for two reasons.

First, proving AI inference functions, where both the server's AI model and client data are in IEEE-754 floating point, requires converting floating-point operations (e.g., addition, multiplication, normalization, rounding) into SNARK-friendly arithmetic or Boolean circuits via bit-level decomposition and explicit rounding logic, which increases circuit complexity. Second, directly hardcoding non-linear functions (e.g. activations) into arithmetic circuits involves creating linear built-ins (e.g., gadgets) for basic non-linear operations (e.g., square root, hyperbolic tangent) and then composing these gadgets to construct arithmetic circuits for complex activation functions. While these constructions maintain high-precision computation, they lead to exponential growth in overall circuit size.

For instance, let y' be a real number produced by earlier linear layers, and the GeLU activation function defined as $\mathbf{GELU}(y') = \frac{y'}{2}\left[1+\mathrm{erf}\left(\frac{y'}{\sqrt{2}}\right)\right] \approx \frac{y'}{2}\left[1+\tanh\left(\sqrt{\frac{2}{\pi}}\,(y'+0.044715\,y'^{3})\right)\right]$. Proving the correctness of this GeLU activation involves handling non-linear operations such as the square root and the hyperbolic tangent (with $\mathbf{tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$). Given the state-of-the-art gadgets for proving the correctness of the square root [20] and tanh [66] functions, and employing the techniques in [67] and [3] that follow the IEEE 754 standard for proving floating-point addition and multiplication operations (with circuit sizes of 2456 and 8854 Boolean gates per addition and multiplication for single precision [67], and 15637 and 44899 Boolean gates for double precision [3]), we estimate that a single GeLU activation requires approximately 1,302,142 R1CS constraints in single precision (and 6,808,760 R1CS constraints in double precision). This overhead increases when multiple GeLU activations are used within a complete AI inference pipeline.

A common strategy to reduce circuit size is to approximate non-linear functions. In particular, a piecewise polynomial approximation $\mathcal{F}^{m,k}$ (as defined in (Equation 1)) partitions the input domain $\mathbf{X}$ into m intervals $\mathcal{I} = \{[t_0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]\}$ and approximates the non-linear function within each interval using a polynomial f^k of degree k. As a result, $\mathcal{F}^{m,k}$ consists of m polynomial pieces $\{f_i^k\}_{i=1}^{m}$, each of degree k, defined over the set of intervals $\mathcal{I}$. Although this approach effectively reduces circuit complexity for complex non-linear operations, it is often insufficient when high precision is required. Specifically, piecewise polynomial approximation introduces an approximation error δ; thus, if these approximated outputs are used in subsequent inference layers, the accumulated errors can lead to considerable precision loss.

Hence, we encounter a *dilemma*: (i) approximation alone cannot meet high-precision demands, and (ii) precisely proving non-linear functions entails large circuit sizes.

4.2 Our Proposed Approach

To address the aforementioned dilemma of proving non-linear functions, we first focus on reducing the circuit size using an approximation strategy. Specifically, we precompute a piecewise polynomial approximation $\mathcal{F}^{m,k}$ for the target non-linear function, which introduces an approximation error δ. Since $\mathcal{F}^{m,k}$ is independent of the client's query, it can be precomputed offline. Given y' as the floating-point output of a previous linear inference layer, the prover selects the appropriate polynomial piece $f_e^k \in \mathcal{F}^{m,k}$ (with $1 \leq e \leq m$) whose domain contains y' and computes $f_e^k(y')$.

Next, to address precision loss by passing forward the approximated output $f_e^k(y')$ into subsequent layers, given y', our strategy is to exactly compute the actual activation function using floating point arithmetics and directly feed this *exact* activation output (y) into the next inference layer. Finally, to prove the correctness of the exact output y, ZIP demonstrates that y is sufficiently close to the approximated output $f_e^k(y')$.

Technical Roadmap. We begin by proving that the polynomial piece f_e^k is correctly selected from the set $\mathcal{F}^{m,k}$.

To achieve this, we first demonstrate the membership of f_e^k in $\mathcal{F}^{m,k}$ by representing the piecewise polynomial $\mathcal{F}^{m,k}$ as a lookup table. Specifically, we embed the private coefficients $\{a_i\}_{i=0}^{k}$ of the selected polynomial piece f_e^k into a private vector $\mathbf{a} = (a_0, a_1, \ldots, a_k)$ of size $k + 1$. Similarly, we represent $\mathcal{F}^{m,k}$ in a public lookup table encoded as a vector $\mathbf{c}$ of size $n = m \cdot (k + 1)$, such that

$$\mathbf{c} = (f_1^k, \ldots, f_e^k, \ldots, f_m^k) = (\{a_i^{(1)}\}_{i=0}^k, \ldots, \{a_i^{(e)}\}_{i=0}^k, \ldots, \{a_i^{(m)}\}_{i=0}^k).$$

Although one could prove that the private elements of $\mathbf{a}$ (resp. f_e^k) are contained in $\mathbf{c}$ (i.e., $\mathcal{F}^{m,k}$) using a standard lookup argument, directly employing existing lookup protocols poses significant challenges. The difficulty arises because the structure of the public table $\mathbf{c}$ for piecewise polynomials does not align directly with the existing protocol designs; specifically, the coefficients of f_e^k (i.e., the elements of $\mathbf{a}$) must appear as an ordered tuple in a designated block within $\mathbf{c}$, whereas standard lookup protocols allow each lookup to reference any individual element at arbitrary positions. Thus, we must ensure that there exists an index $i \in \{1, \ldots, m\}$ for each $j \in \{0, \ldots, k\}$ such that $\mathbf{a}^{(i)}[j] = \mathbf{c}[(i - 1)(k + 1) + j]$.

Next, to complete the lookup proof of membership, we prove that the coefficients of the selected polynomial piece f_e^k are taken from the appropriate row of $\mathcal{F}^{m,k}$ corresponding to the interval $\mathcal{I} = \{[t_0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]\}$ in which y' resides. While a standard range proof could verify that y' lies in the correct interval for the selected f_e^k, directly applying existing range-proof protocols poses challenges. Similar to the earlier case, the structure of the public table $\mathbf{c}$ and the range check requirements do not directly correspond to typical designs: we must keep the selected interval *private* during the proof while also proving that this private interval *indeed* belongs to the set $\mathcal{I}$.

Without enforcing these structures, a malicious prover could produce a proof using incorrect coefficients or intervals, thereby compromising the overall proof correctness.

Once the correct polynomial f_e^k is selected, we prove the correctness of the actual non-linear function output y by introducing an approximation relation technique grounded in numerical analysis [27, 59]. Using this relative error–based technique, ZIP allows the (honest) prover to guarantee that the approximated computation output (i.e., polynomial evaluation $f_e^k(y')$) is sufficiently close to the precise output y, with an error bound δ such that

$$|y - f_e^k(y')| \leq \delta |f_e^k(y')|. \tag{2}$$

Hence, to address the efficiency and precision challenges outlined above in proving non-linear functions, we proceed as follows:

(1) We extend lookup arguments to securely prove that $f_e^k \in \mathcal{F}^{m,k}$.
(2) We adapt range proofs to prove that f_e^k is selected from the correct interval that contains y'.
(3) We prove that the relation (2) holds.

Remark. Efficiently proving non-linear activation functions using the approximation relation technique (2) allows a malicious prover to deviate from selecting the correct activation output y, since the zero-knowledge setting prevents the verifier from directly accessing y. However, ZIP limits the prover's malicious behavior by enforcing a small error bound relative to the target approximated polynomial $f_e^k(y')$, thereby preventing arbitrary selection of the activation output.

4.3 High-Precision Approximation Arguments

In this section, we detail the three key components of our high-precision zero knowledge proof for non-linear functions. First, we extend the Caulk lookup argument to enforce that the prover selects the correct ordered tuple of polynomial coefficients from a public table. Second, we introduce a private interval lookup argument that proves the prover selects the correct interval corresponding to the value's range, without revealing the interval itself. Finally, we describe our approximation-relation technique to prove that the evaluated polynomial output is sufficiently close to the precise non-linear function output within the specified error bound.

4.3.1 Polynomial Coefficients Lookup. In this section, we extend the Caulk [71] lookup argument protocol to securely prove that f_e^k is a valid polynomial piece of $\mathcal{F}^{m,k}$. We start by giving a high-level design of Caulk. We then explain why applying it directly to our setting is inadequate, as it does not enforce the ordering and structural constraints required for correct polynomial selection. To overcome this, we introduce a set of additional arithmetic constraints that extend Caulk to support ordered tuple extraction from a designated block of the public table.

Caulk Overview. Caulk lets a prover demonstrate that every element of committed private vector $\mathbf{a}$ of size $k + 1$ appears in a committed public table $\mathbf{c}$ of size n. Accordingly, it first constructs a subvector $\mathbf{c}_I$ of $\mathbf{c}$ that contains all elements of $\mathbf{c}$ which appear in $\mathbf{a}$ (without duplicates, i.e., $\mathbf{c}[i] = \mathbf{a}[j]$). Let $\mathbb{V}_k = \{1, \nu, \ldots, \nu^{k-1}\}$ with $\nu^k = 1$, and $\mathbb{H} = \{1, \omega, \ldots \omega^{n-1}\}$ with $\omega^n = 1$ be subgroups of roots of unity. Define the subset $\mathbb{H}_I = \{\omega^{i-1}\}_{i \in I}$. Furthermore, let $\mu_j(x)_{j=0}^k$, $\lambda_i(x)_{i=0}^n$, and $\tau_i(x)_{i \in I}$ be the Lagrange interpolation polynomials defined over these sets of roots of unity. It then encodes the vectors $\mathbf{a}$, $\mathbf{c}$, and $\mathbf{c}_I$ into the polynomials $A(x) = \sum_{j=0}^k a_j \mu_j(x)$, $C(x) = \sum_{i=0}^n c_i \lambda_i(x)$, and $C_I(X) = \sum_{i \in I} c_i \tau_i(x)$, and defines an auxiliary polynomial $U(x) = \sum_{j=0}^k w^{i_j} \mu_j(x)$, and committing to all these polynomials.

Let $Z_{V_k}(x)$ be the vanishing polynomial for $\mathbb{V}_k$ and let $Z_I(x)$ be the vanishing polynomial for the interpolation nodes in $C_I(x)$. To prove that all elements of $C_I(x)$ appear somewhere in $C(x)$, it employs KZG proofs of openings to show that

$$C(x) - C_I(x) \;=\; Z_I(x)\, H_1(x) \tag{3}$$

for some $H_1(x)$. Because $C_I(X)$, $Z_I(X)$, and $U(x)$ could potentially reveal private information about the elements of $\mathbf{a}$ (or their corresponding indices in $\mathbf{c}$), it introduces additional blinding factors to $C_I(X)$, $Z_I(X)$, and $U(x)$. Next, Caulk proves the correct formation of the blinded $Z_I(x)$ by

$$Z_I(U(x)) = Z_{V_k}(x) H_2(x) \tag{4}$$

for some $H_2(x)$. Finally, after proving the well-formation of blinded polynomial $U(x)$, it proves that the commitment to $C_I(x)$ indeed matches the same committed values in by

$$C_I(U(x)) - A(x) = Z_{V_k}(x) H_3(x) \tag{5}$$

for some $H_3(x)$.

Limitations of Caulk for Piecewise Polynomial Coefficients Lookup. In ZIP, the elements of the private vector $\mathbf{a}$ (1) must form a single *ordered tuple* in $\mathbf{c}$, and (2) reside entirely within exactly a *designated* blocks of size $k + 1$ in the public $\mathbf{c}$. However, directly employing the original Caulk protocol in the ZIP setting (i.e., where the table contains a set of polynomial coefficients) introduces a critical problem. This is because Caulk protocol (following Equations (3)-(5)) does not enforce any ordering constraints on the elements in $\mathbf{a}$ and allows the elements of $\mathbf{a}$ to be taken from arbitrary positions in $\mathbf{c}$.

Specifically, consider a public table $\mathbf{c}$ composed of m vectors (i.e., blocks) $\{\mathbf{a}^{(i)}\}_{i=1}^{m}$, each of size $k+1$, such that

$$\begin{aligned}\mathbf{c} &= (\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(e)}, \ldots, \mathbf{a}^{(m)}) \\ &= \big(\underline{a_0^{(1)}, \ldots, a_k^{(1)}}, \ldots, \underline{a_0^{(e)}, \ldots, a_k^{(e)}}, \ldots, \underline{a_0^{(m)}, \ldots, a_k^{(m)}}\big).\end{aligned} \tag{6}$$

For each block $i \in \{1, \ldots, m\}$, the valid indices corresponding to the elements of the private vector $\mathbf{a}^{(i)}$ in $\mathbf{c}$ lies in the index range

$$[(i-1)(k+1), i(k+1)-1]. \tag{7}$$

Hence, if the selected private vector is $\mathbf{a}^{(e)}$ (with $1 \leq e \leq m$) with elements $\{a_i^{(e)}\}_{i=0}^{k}$, then the correct ordered tuple for $\mathbf{a}$ is

$$\mathbf{a} = (a_0^{(e)}, a_1^{(e)}, \ldots, a_k^{(e)}). \tag{8}$$

However, if a malicious prover constructs an invalid tuple from arbitrary indices in $\mathbf{c}$ such that $\hat{\mathbf{a}} = (a_2^{(e-1)}, a_k^{(m)}, \ldots, a_0^{(e)})$ or forms a seemingly ordered tuple spanning two blocks, such that $\hat{\mathbf{a}} = (a_k^{(e-1)}, a_0^{(e)}, \ldots, a_{k-1}^{(e)})$, then, by following the original Caulk protocol, the malicious prover could exploit the lack of ordering constraints and produce a proof that the verifier would accept; thus, compromising the overall proof's integrity.

Our Proposed Zero-Knowledge Ordered Tuple Arguments for Piecewise Polynomial Coefficients Lookup. To enable proving ordered tuple selection from designated blocks, ZIP extends Caulk with new arithmetic constraints that (i) confine all $k+1$ elements of private vector $\mathbf{a}$ to exactly one *predetermined* block (defined in (7)) of $\mathbf{c}$, (ii) ensure the elements of $\mathbf{a}$ are in consecutive positions within the selected block with their order preserved, and (iii) leak no additional information about which block is chosen.

We first define $k+1$ auxiliary indicator vectors $\{\mathbf{s}_j\}_{j=0}^{k}$, each of size n. Given the public table $\mathbf{c}$ of size n (cf. (6)), and the private vector $\mathbf{a}^{(e)}$ of size $k+1$ (cf. (8)), we construct $\mathbf{s}_j$ for each $j \in \{0, \ldots, k\}$ as follows

$$\mathbf{s}_j[i] = \begin{cases} 1, & \text{if } i = (e-1)(k+1)+j, \\ 0, & \text{otherwise.} \end{cases}$$

More precisely, each $\mathbf{s}_j$ acts as an indicator vector that marks the position of $\mathbf{a}^{(e)}[j]$ in $\mathbf{c}$. We then commit to these auxiliary vectors $\mathbf{s}_j$ and introduce new constraints to prove the well-formation of $\mathbf{s}_j$ constructed by the prover.

Accordingly, for each $j \in \{0, \ldots, k\}$, we impose the constraints

$$\sum_{i=0}^{n-1} \mathbf{s}_j[i] = 1 \tag{9}$$

These constraints ensure that each $\mathbf{s}_j$ has exactly one nonzero entry, thereby selecting exactly one position in $\mathbf{c}$. Subsequently, for every $j \in \{0, \ldots, k\}$ and $i \in \{0, \ldots, n-1\}$, we set constraints

$$\mathbf{s}_j[i](\mathbf{s}_j[i]-1) = 0 \tag{10}$$

this enforce that each $\mathbf{s}_j[i]$ is binary (i.e. $\mathbf{s}_j[i] \in \{0,1\}$). Moreover, we introduce ordering constraints. Particularly, for every $i \in \{1, \ldots, m\}$ and every $j \in \{1, \ldots, k\}$ we require

$$\mathbf{s}_j[(i-1)(k+1)+j] = \mathbf{s}_{j-1}[(i-1)(k+1)+(j-1)]. \tag{11}$$

These constraints link the positions of the nonzero entries between auxiliary vectors $\mathbf{s}_j$, where they guarantee that the positions of their nonzero entries are adjacent within the designated block; thus, the elements $(a_0^{(e)}, \ldots, a_k^{(e)})$ of the private vector $\mathbf{a}$, appear in a contiguous order in c. After that, we impose a single-block selection constraint

$$\sum_{i=1}^{m} \sum_{j=0}^{k} \mathbf{s}_j[(i-1)(k+1)+j] = (k+1). \tag{12}$$

Since each $\mathbf{s}_j$ has exactly one 1, any single block of length $(k+1)$ will contain exactly one entry of 1 from each $\mathbf{s}_j$. Thus, the only way the global sum can equal $(k+1)$ is if exactly one block contains all the nonzero entries, while the others remain zero. Then, we enforce

$$\mathbf{s}_0[n-1] = 0 \text{ and } \mathbf{s}_k[0] = 0, \tag{13}$$

which ensures the selected contiguous positions do not spill over from the end of $\mathbf{c}$ back to the beginning. Finally, for each $j \in \{0, \ldots, k\}$,

$$\mathbf{a}[j] = \sum_{i=0}^{n-1} \mathbf{s}_j[i]\mathbf{c}[i] \tag{14}$$

These constraints confirm that the single nonzero element 1 in $\mathbf{s}_j$, correctly indicates the selected index in $\mathbf{c}$ and truly corresponds to the jth element of $\mathbf{a}$ (i.e., $\mathbf{a}^{(e)}[j]$).

Given the piecewise polynomial approximation $\mathcal{F}^{m,k}$ of the target non-linear function, ZIP embeds the coefficients of all the polynomial pieces into a public table $\mathbf{c}$ of size $n = m \cdot (k+1)$ s.t.

$$\begin{aligned}\mathbf{c} &= (f_1^k, \ldots, f_e^k, \ldots, f_m^k) \\ &= \big(\underline{a_0^{(1)}, \ldots, a_k^{(1)}}, \ldots, \underline{a_0^{(e)}, \ldots, a_k^{(e)}}, \ldots, \underline{a_0^{(m)}, \ldots, a_k^{(m)}}\big).\end{aligned}$$

Given the selected polynomial piece f_e^k (with $1 \leq e \leq m$) having coefficients $\{a_i^{(e)}\}_{i=0}^{k}$, we form the corresponding private vector as $\mathbf{a} = (a_0^{(e)}, a_1^{(e)}, \ldots, a_k^{(e)})$. Finally, by enforcing our proposed arithmetic constraints (e.g., Equations (9)-(14)), ZIP securely proves that $f_e^k \in \mathcal{F}^{m,k}$.

Lemma 1 (Soundness of Extended Lookup Argument). *Any prover that satisfies constraints* (9)–(14) *must have chosen a single block of* $k+1$ *consecutive coefficients in* $\mathbf{c}$ *and in the correct order. In particular, the private vector* $\mathbf{a}$ *equals exactly one of the valid blocks in* $\mathbf{c}$ *(Equation 6).*

Proof. Suppose, for the sake of contradiction, that a malicious prover $\mathcal{P}^*$ satisfies constraints (9)–(14) but the private vector $\mathbf{a}$ does *not* match with any single contiguous block of length $k+1$ in the public table $\mathbf{c}$. We consider the only ways this can happen and show each contradicts one of the proposed constraints.

- **Scenario 1: Some coefficient lies outside c.** If $\mathcal{P}^*$ attempts to include any $\mathbf{a}[j] \notin \{\mathbf{c}[0], \ldots, \mathbf{c}[n-1]\}$, then constraint (14) would force a contradiction, where the right-hand side is a convex combination of entries in $\mathbf{c}$, so it cannot produce a value outside

c. Formally, by the soundness of back-end lookup proof, such an event can occur only with negligible probability.

- **Scenario 2: All $\mathbf{a}[j]$ lie in c, but not in a single ordered block.** Since each $\mathbf{s}_j$ selects exactly one position ((9)–(10)), and there are $k+1$ vectors, exactly $k+1$ entries of **c** are chosen. We break this scenario into three subcases:
  - **Case 1: Out-of-order selection.** If the prover picks $k+1$ entries all from the same block but not in increasing index order, then constraint (11) is violated. This constraint forces the one in $\mathbf{s}_j$ to be immediately adjacent to the one in $\mathbf{s}_{j-1}$, preventing any reordering.
  - **Case 2: Mixing multiple blocks.** If the prover's $k+1$ selected entries span more than one block, then the block-sum constraint (12) cannot hold; it requires that all $k+1$ ones lie in a single block of length $k+1$, which is impossible if they are drawn from different blocks.
  - **Case 3: Wrap-around selection.** If the prover attempts to wrap a block across the end and beginning of **c**, selecting entries from both the final and initial positions, then constraint (13) is violated. This constraint prevents placing the first or last selected coefficient across the public table boundary.

In all cases, at least one of the enforced constraints is contradicted. Therefore the only way to satisfy (9)–(14) is to choose exactly one contiguous block of length $k+1$, in the correct order. Constraint (14) then ensures that the private vector **a** matches precisely those $k+1$ entries from **c**. This completes the proof. □

4.3.2 Private Interval Lookup Arguments. In this section, we adapt standard range proofs to prove that the private value y', produced by preceding linear layers, lies within the correct interval of $\mathcal{I} = \{[t_0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]\}$, even in the presence of a malicious prover. We first highlight why standard range proofs are insufficient in our setting, as they may leak the selected interval or allow invalid interval selection. To address this, we introduce a set of arithmetic constraints that privately enforce the ordered selection of valid interval endpoints from a public table. We then combine this with a floating-point range check to prove that y' lies within the selected interval, without revealing which interval was chosen.

Issues of Using Standard Range Proofs for Private Interval. Given y', ZIP selects the polynomial piece f_e^k from $\mathcal{F}^{m,k}$, if and only if y' lies in the interval $[t_{e-1}, t_e] \in \mathcal{I}$ for some $1 \le e \le m$. Although standard range-proof protocols can be used to prove $t_{e-1} \le y' \le t_e$, they face two challenges.

(1) First, directly applying the standard range proof might reveal the selected interval $[t_{e-1}, t_e]$ during the proof process. Although the piecewise polynomial $\mathcal{F}^{m,k}$ and its intervals $\mathcal{I}$ are public, revealing the specific interval $[t_{e-1}, t_e]$ would compromise the privacy of the coefficients of f_e^k.
(2) Second, preserving the confidentiality of the selected interval requires additional proof that the chosen interval $[t_{e-1}, t_e]$ is indeed one of the valid intervals in $\mathcal{I}$. Without this additional proof, a malicious prover could arbitrarily select an interval and bypass the proof.

Our Proposed Zero-Knowledge Private Interval Arguments. To overcome both challenges, we must simultaneously (i) hide the chosen interval, (ii) prove that its two endpoints are adjacent in the public set $\mathcal{I}$, and (iii) prove that y' lies within those hidden bounds. Accordingly, to keep the selected interval $[t_{e-1}, t_e]$ private, we first construct a public table $\mathbf{c}'$ of size $m+1$ that contains all interval endpoints from $\mathcal{I}$ such that

$$\mathbf{c}' = (t_0, \ldots, t_{e-1}, t_e, \ldots, t_m). \tag{15}$$

We also define a private vector $\mathbf{a}'$ of size $k' = 2$ representing the interval endpoints containing y', which we aim to keep private such that

$$\mathbf{a}' = (t_{e-1}, t_e). \tag{16}$$

Next, we employ the lookup argument to prove that all elements of committed $\mathbf{a}'$ are contained in a committed public table $\mathbf{c}'$. However, in ZIP the elements of $\mathbf{a}'$ (i.e., (t_{e-1}, t_e)) must appear as a single *ordered tuple* within $\mathbf{c}'$. Since the standard index-item correspondence lookup argument design does not enforce ordering constraints (i.e., each element is treated independently), naively employing a lookup argument leaves our ZIP vulnerable to a malicious prover.

Our Proposed Ordered Interval Arguments. To prevent a malicious prover from bypassing the proof with non-consecutive or misordered endpoints, we introduce new additional arithmetic constraints that enforce correct ordering which guarantees that the elements of $\mathbf{a}'$ (i.e., (t_{e-1}, t_e)) appear as a correctly *ordered tuple* in $\mathbf{c}'$ thereby preventing arbitrary selection. To this end, we introduce $k' = 2$ auxiliary vectors $\{\mathbf{s}'_j\}_{j=0}^{k'-1}$, each of size $m+1$ (with arbitrary m, but k' fixed at 2). Given the public table $\mathbf{c}'$ (as in (15)) and the private vector $\mathbf{a}'$ (as in (16)) corresponding to the selected interval $[t_{e-1}, t_e]$ for f_e^k, for each $j \in \{0, 1\}$, we set each auxiliary $\mathbf{s}'_j$ s.t.

$$\mathbf{s}'_j[i] = \begin{cases} 1, & \text{if } i = e + j - 1, \\ 0, & \text{otherwise.} \end{cases}$$

Each auxiliary vector $\mathbf{s}'_j$ serves as an indicator that pinpoints the exact position of the jth endpoint (i.e., t_{e-1} for $j = 0$ and t_e for $j = 1$) in $\mathbf{c}'$. We then commit to these auxiliary vectors $\mathbf{s}'_j$ and prove the well-formation of the $\mathbf{s}'_j$. To do so, for each $j \in \{0, 1\}$, we propose the following constraints

$$\sum_{i=0}^{m} \mathbf{s}'_j[i] = 1 \tag{17}$$

These constraints ensure each $\mathbf{s}'_j$ has exactly one nonzero entry. Then, for every $j \in \{0, 1\}$ and $i \in \{0, \ldots, m\}$, we provide constraints

$$\mathbf{s}'_j[i](\mathbf{s}'_j[i] - 1) = 0 \tag{18}$$

which guarantee that each entry of $\mathbf{s}'_j$ is binary. Next, we propose the ordering constraints such that for each $i \in \{1, \ldots, m\}$ and $j \in \{1\}$ (i.e., $j = 1$ since we have two vectors), we enforce

$$\mathbf{s}'_j[i + j - 1] = \mathbf{s}'_{j-1}[i + j - 2], \tag{19}$$

or $\mathbf{s}'_1 i = \mathbf{s}'_0[i-1]$. This constraint ensures that if the first endpoint t_{e-1} is at position $(i-1)$, then the second endpoint t_e must be right next to it (i.e., at position i); thus, $[t_{e-1}, t_e]$ appear consecutively in $\mathbf{c}'$. Then, we impose

$$\mathbf{s}'_0[m] = 0 \text{ and } \mathbf{s}_1[0] = 0, \tag{20}$$

which ensures not to select an endpoint at the very end of $\mathbf{c}'$ (for $j = 0$) together with one at the very beginning (for $j = 1$). Finally, for every $j \in \{0, 1\}$, we provide

$$\mathbf{a}'[j] = \sum_{i=0}^{m} \mathbf{s}'_j[i]\mathbf{c}'[i] \tag{21}$$

which confirm that the nonzero entry in $\mathbf{s}'_j$ select the correct ordered internal tuple $[t_{e-1}, t_e]$ in $\mathbf{a}'$ from $\mathbf{c}'$.

After imposing additional arithmetic constraints to correctly select the valid interval $[t_{e-1}, t_e]$ from $\mathcal{I}$, we employ a range proof to prove the secrete value y' lies within the chosen interval i.e.

$$t_{e-1} \le y' \le t_e. \tag{22}$$

To achieve this, since the selected interval $[t_{e-1}, t_e]$ are kept private, we first introduce two auxiliary witnesses z_1, z_2 such that $z_1 = y' - t_{e-1}$ and $z_2 = t_e - y'$. Subsequently, to prove Equation (22), we prove that z_1 and z_2 are non negative, which implies y' is indeed inside the correct interval. Since z_1, z_2 are floating point numbers presented as a tuple $(\hat{s}, \hat{e}, \hat{m})$, to detect they are non-negative, we can check their sign bit $\hat{s}$ and to check if they are 0, we check their mantissa $\hat{m}$ such that $(\hat{s}_{z_1} = 0 \vee \hat{m}_{z_1} = 0) = 1$, and $(\hat{s}_{z_2} = 0 \vee \hat{m}_{z_2} = 0) = 1$. Together, these constraints prove the correctness of $t_{e-1} \le y' \le t_e$.

Lemma 2 (Soundness of Private Interval Argument). *Any prover that satisfies constraints* (17)–(21) *along with the non-negativity checks on* z_1 *and* z_2 *must have selected two consecutive endpoints* (t_{e-1}, t_e) *in* $\mathbf{c}'$ *and shown* $t_{e-1} \le y' \le t_e$.

Proof. Assume toward contradiction that a malicious prover $\mathcal{P}^*$ satisfies constraints (17)–(21) and the non-negativity checks on z_1 and z_2, yet fails to both (a) select two *consecutive* endpoints (t_{e-1}, t_e) from $\mathbf{c}'$, or (b) prove $t_{e-1} \le y' \le t_e$. We examine the possible cases and show each violates one of the enforced constraints.

- **Scenario 1: Endpoints not in $\mathbf{c}'$.** If t_{e-1} or t_e is not in the public table $\mathbf{c}'$, then constraint (21) cannot hold, where the right-hand side is a convex combination of entries in $\mathbf{c}'$, so it cannot produce a value outside $\mathbf{c}'$. By the soundness of the underlying lookup argument, this event has only negligible probability.
- **Scenario 2: Non-consecutive or misordered endpoints.** Each $\mathbf{s}'_j$ selects exactly one index (by (17)-(18)), and there are two such vectors. If these two ones do not occupy adjacent positions in $\mathbf{c}'$, then one of the following must occur:
  - **Case 1: Out-of-order selection.** The selected indices are not in increasing order. Constraint (19) enforces that the "1" in $\mathbf{s}'_1$ immediately follows the "1" in $\mathbf{s}'_0$. Any violation contradicts this adjacency requirement.
  - **Case 2: Wrap-around selection.** The prover picks one endpoint at the last position of $\mathbf{c}'$ and the other at the first position. Constraint (20) prevents exactly that configuration.

  Hence the only way to satisfy (17)–(21) is to pick two *adjacent* entries in $\mathbf{c}'$.
- **Scenario 3: Failing the range check.** Even with correct consecutive endpoints, the prover must still prove (22). We set $z_1 = y' - t_{e-1}$ and $z_2 = t_e - y'$, and enforce non-negativity of each z_i by checking its IEEE-754 sign bit and mantissa. If $y' < t_{e-1}$, then $z_1 < 0$ and its sign bit test fails. If $y' > t_e$, then $z_2 < 0$ and its test fails. Thus any proof passing the non-negativity checks must satisfy (22).

Therefore, the only way to satisfy all constraints is to (1) select two consecutive entries (t_{e-1}, t_e) from $\mathbf{c}'$ and (2) prove $t_{e-1} \le y' \le t_e$. This completes the proof. □

4.3.3 Proving Approximation Correctness Relation. This section details how ZIP proves $|y - f_e^k(y')| \le \delta|f_e^k(y')|$, thereby showing that the approximated output $f_e^k(y')$ is within the permitted relative error δ of the exact value y. This, in turn, restricts any potential malicious behavior by the prover to a small, quantifiable error bound.

Proving the Polynomial Evaluation $f_e^k(y')$. Let y' and the private coefficient vector $\mathbf{a}^{(e)} = (a_0^{(e)}, a_1^{(e)}, \dots, a_k^{(e)})$ be given in IEEE-754 format. A naïve term-by-term proof of $f_e^k(y') = a_k^{(e)} y'^k + \cdots + a_1^{(e)} y' + a_0^{(e)}$ requires $2k - 1$ floating-point multiplications and k floating-point additions. To reduce this overhead, ZIP applies the Horner's method [32]. Specifically, we re-write the polynomial $f_e^k(y')$ in nested form $f_e^k(y') = \mathbf{a}_0^{(e)} + y'(\mathbf{a}_1^{(e)} + y'(\cdots + y'\mathbf{a}_k^{(e)}))$, which costs only k floating multiplications and k floating point additions. Next, we introduce two auxiliary witnesses l_1, l_2 such that $l_1 = y - f_e^k(y')$ and $l_2 = \delta \cdot f_e^k(y')$.

Enforcing the inequality. ZIP proves the inequality $|l_1| \le |l_2|$ which is equivalent to $l_1^2 \le l_2^2$, implying $(l_1 + l_2)(l_2 - l_1) \ge 0$. Thus, we introduce auxiliary witness z_3 such that $z_3 = (l_1 + l_2)(l_2 - l_1)$. As a result, the prover is required to prove z_3 is non-negative. Because z_3 is represented in floating-point form $(\hat{s}, \hat{e}, \hat{m})$, non-negativity can be proved by $(\hat{s}_{z_3} = 0 \vee \hat{m}_{z_3} = 0) = 1$. Together, these constraints prove the approximation correctness relation $|y - f_e^k(y')| \le \delta|f_e^k(y')|$.

4.4 Realizing Zero-Knowledge AI Inference

After the server commits to its pre-trained model $\mathbf{W}$ and receives a client's input $\mathbf{X}$, it performs AI inference and returns the inference result with a zero-knowledge proof of correct computation. Below, we describe how each algorithm in our ZIP protocol operates.

- $\underline{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, l)}$: Given a security parameter λ and an upper bound l on model size, this algorithm constructs the public parameters pp. Given the activation function Act, it invokes [21] to interpolate the piecewise polynomial approximation $\mathcal{F}^{m,k}$, defined over the set of intervals I, with approximation error δ. The lookup tables $\mathbf{c}$ and $\mathbf{c}'$ are then derived from $\mathcal{F}^{m,k}$. Next, the algorithm runs the CP-SNARK setup to obtain $\mathsf{pp}' \leftarrow \mathsf{CPZKP.Setup}(1^\lambda)$. Finally, it outputs the public parameters as $\mathsf{pp} = (\mathsf{pp}', \mathbf{c}, \mathbf{c}', m, k, \delta)$.
- $\underline{\hat{\mathsf{cm}} \leftarrow \mathsf{Comm}(\mathbf{W}, r, \mathsf{pp})}$: Given randomness r, it commits to $\mathbf{W}$ by invoking $\hat{\mathsf{cm}} \leftarrow \mathsf{CPZKP.Comm}(\mathbf{W}, r, \mathsf{pp})$, and returns cm.
- $\underline{(\pi, \mathrm{y}) \leftarrow \mathsf{Infer}(\mathbf{W}, \mathbf{X}, \mathsf{pp})}$: Upon receiving query data $\mathbf{X}$, it executes the inference pipeline locally using the committed model $\mathbf{W}$. Since $\mathbf{W}$ and $\mathbf{X}$ are in IEEE-754 format, it executes techniques in §4.1-§4.3 to produce arithmetic constraints ϕ_{act} for non-linear layers (e.g., activations). It then adopts the method of [20] to generates arithmetic constraints ϕ_{cnv}, ϕ_{pl}, and ϕ_{fc} for all linear operations (e.g., convolution, pooling, fully connected). Next, it commits to auxiliary witnesses aux (e.g., layer outputs and activations) via $\mathsf{cm}' \leftarrow \mathsf{CPZKP.Comm}(\mathsf{aux}, r_2, \mathsf{pp})$, and produces a proof by calling $\pi' \leftarrow \mathsf{CPZKP.Prov}(\phi, \mathbf{W}, \mathsf{pp})$, where ϕ is the

merge arithmetic circuit of ϕ_{act}, ϕ_{cnv}, ϕ_{pl}, ϕ_{fc}. Finally, it returns the inference output $\mathbf{y}$ with proof $\pi := (\pi', \mathsf{cm}')$.

- $\{0,1\} \leftarrow \mathsf{Ver}(\mathbf{y}, \mathsf{cm}, \pi, \mathsf{pp})$: Given the inference result $\mathbf{y}$, a commitment cm, and a proof π, it parses $\pi := (\pi', \mathsf{cm}')$ and then invokes $b \leftarrow \mathsf{CPZKP.Ver}(\pi, \mathsf{cm}, \mathbf{y}, \mathsf{pp})$, where $\mathsf{cm} = (\hat{\mathsf{cm}}, \mathsf{cm}')$.

Security. We state the security of ZIP as follows.

Theorem 1 (Security of ZIP). *Assuming the back-end CP-SNARK is secure (per Definition 1), and both the extended lookup argument and the adapted range proof (based on Caulk [71]) are sound and zero-knowledge, then* ZIP *is a secure zero-knowledge ML inference scheme as defined in Definition 2.*

Proof. We establish the completeness, soundness, and zero knowledge properties followed by composing the security of the back-end CP-SNARK with Lemmas 1 and 2.

Completeness. In the ZIP protocol, if the server honestly computes ML inference using model $\mathbf{W}$ on client input $\mathbf{X}$ to produce output $\mathbf{y}$, the verifier accepts with probability 1. Correctness of y in our scheme follows from completeness of the back-end CP-SNARK, the extended lookup argument, and the adapted range proof. Specifically, by Lemma 1, the prover can satisfy the extended lookup constraints by selecting a valid, ordered block of coefficients from the public polynomial table. By Lemma 2, the prover can also satisfy the adapted range proof constraints by selecting the correct interval (t_{e-1}, t_e) and proving that $t_{e-1} \le y' \le t_e$.

Soundness. Suppose a malicious server $\mathcal{P}^*$ produces an accepting proof for $(\mathbf{X}, \mathbf{y})$ with $\mathbf{y} \neq \hat{\mathcal{F}}(\mathbf{W}, \mathbf{X})$. By knowledge-soundness of the back-end CP-SNARK (Definition 1), an extractor exists that recovers a witness satisfying all circuit constraints. In particular:

- By Lemma 1, the extracted lookup witness must correspond to exactly one contiguous tuple and valid block of elements in $\mathbf{c}$.
- By Lemma 2, the extracted interval witness must be two consecutive endpoints (t_{e-1}, t_e) satisfying $t_{e-1} \le y' \le t_e$.
- The remaining arithmetic constraints then force the correct evaluation of both linear and non-linear layers (including the relative error check), so the extracted witness computes $\mathbf{y} = \hat{\mathcal{F}}(\mathbf{W}', \mathbf{X})$ for the extracted model parameters $\mathbf{W}'$.

Hence we obtain $\mathbf{y} = \hat{\mathcal{F}}(\mathbf{W}', \mathbf{X})$, contradicting $\mathbf{y} \neq \hat{\mathcal{F}}(\mathbf{W}, \mathbf{X})$. The only possibility would be $\mathbf{W}' \neq \mathbf{W}$ yet $\hat{\mathcal{F}}(\mathbf{W}', \mathbf{X}) = \mathbf{y}$, implying that $\mathcal{P}^*$ extracted a different model $\mathbf{W}'$ still satisfying the proof—contradicting the binding of the commitment scheme. Thus, no polynomial-time adversary can produce a false accepting proof except with negligible probability.

Zero-Knowledge. We show that a semi-honest verifier's view can be simulated using only the public inputs $(\mathbf{X}, y)$. Let $\mathcal{S}$ be the simulator that proceeds via three hybrids:

- **Hybrid 0 (Real).** Run the real ZIP protocol: (1) generate the extended lookup argument and adapted range proof, and (2) produce the CP-SNARK proof over the full arithmetic circuit, which includes floating-point operations, polynomial approximation, indicator-vector logic, and the approximation check.
- **Hybrid 1.** Replace the extended lookup and range proofs with simulated proofs that preserve zero-knowledge. Specifically, the lookup proof is simulated using the position-hiding simulator for Caulk, along with simulated auxiliary vectors that satisfy our ordering and block structure constraints. The range proof is simulated via the arithmetic constraints defined in Lemma 2, which preserve the privacy of the selected interval. Since our construction ensures that no information about the selected polynomial piece or interval is leaked to the verifier, the simulated view is indistinguishable from Hybrid 0.
- **Hybrid 2 (Ideal).** Replace the CP-SNARK proof with its zero knowledge simulator. By the zero knowledge property of the back-end CP-SNARK (Definition 1), the verifier's view remains indistinguishable from Hybrid 1.

Since each hybrid transition modifies exactly one component and remains computationally indistinguishable, the real transcript (Hybrid 0) is indistinguishable from the fully simulated one (Hybrid 2). Therefore, the verifier learns nothing beyond $(\mathbf{X}, y)$, completing the zero-knowledge proof. □

5 Experimental Evaluation

5.1 Implementation

We fully implemented ZIP in about 11,000 lines of Python, Go, and Rust. We used the NFGen library [21, 22] to generate high-accuracy piecewise-polynomial approximations for non-linear activations, including GeLU, SeLU, and ELU, as well as other non-linear functions in Softmax and LayerNorm, such as $\exp(x) = e^x$ and $1/\sqrt{x}$. We integrated the Rust-based Caulk library [72] for multi-lookup arguments and compiled our arithmetic circuits in Go using the gnark library [6] with the PlonK CP-SNARK protocol [25].

However, the original PlonK protocol [25] initiates proving immediately after committing to model parameters $\mathbf{W}$. Since ZIP requires the server to commit to $\mathbf{W}$ in advance and broadcasts the commitment, we adapted PlonK by decoupling the commitment phase from the proof generation step. Furthermore, we replaced the original KZG PC scheme in PlonK, which is computationally hiding under the discrete logarithm assumption, with the stronger commitment scheme $\mathsf{PolyCommit}_{\mathsf{Ped}}$ introduced in [35]. This new scheme achieves unconditional hiding under the Strong Diffie-Hellman (SDH) assumption, preventing disclosure of private model parameters from the broadcast commitment, especially when parameters are small enough to be brute-forced.

We also reimplemented the LeNet-5 [36] and mini-BERT [60] inference models to extract all required witnesses for the commit-and-prove protocol. Our full implementation is publicly available at: https://github.com/vt-asaplab/ZIP

5.2 Configuration

Hardware. We used a server with a 48-core Intel(R) Xeon(R) Platinum 8360Y CPU (2.40 GHz) and 512 GB RAM.

System Parameters. We performed all computations in ZIP using IEEE-754 double-precision (64-bit) floating point. We interpolated piecewise-polynomial approximations of ELU[2], SeLU[3], and GeLU[4] by sampling each function at 1,000 points over the domains

[2] $\mathrm{ELU}_{\alpha}(x) = \begin{cases} x, & x \ge 0 \\ \alpha\,(e^x - 1), & x < 0 \end{cases}$

[3] $\mathrm{SeLU}_{\lambda,\alpha}(x) = \lambda \begin{cases} x, & x \ge 0 \\ \alpha\,(e^x - 1), & x < 0 \end{cases}$

[4] $\mathrm{GeLU}(x) = \frac{x}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \approx \frac{x}{2}\left[1 + \tanh\left(\sqrt{\frac{2}{\pi}}\,(x + 0.044715\,x^3)\right)\right]$

Table 1: Number of R1CS Constraints for IEEE-754 Double-Precision GeLU, SeLU, and ELU: ZIP vs. Baseline (w/AG).

# Acts	GeLU			SeLU			ELU		
	ZIP	Baseline (w/AG)	|C| Gain†	ZIP	Baseline (w/AG)	|C| Gain†	ZIP	Baseline (w/AG)	|C| Gain†
2^0	5.8×10^3	6.8×10^6	**1172×**	3.7×10^3	2.0×10^6	**545×**	3.3×10^3	1.9×10^6	**569×**
2^4	8.2×10^4	1.1×10^8	**1326×**	5.0×10^4	3.2×10^7	**634×**	4.5×10^4	3.0×10^7	**664×**
2^8	1.2×10^6	1.7×10^9	**1403×**	7.4×10^5	5.1×10^8	**687×**	6.8×10^5	4.8×10^8	**714×**
2^{12}	1.9×10^7	2.8×10^{10}	**1477×**	1.1×10^7	8.1×10^9	**743×**	9.8×10^6	7.7×10^9	**786×**
2^{16}	3.0×10^8	4.5×10^{11}	**1482×**	1.7×10^8	1.3×10^{11}	**748×**	1.6×10^8	1.2×10^{11}	**791×**

† "|C| Gain" indicates how many times smaller ZIP circuit is compared to Baseline (w/AG).

Table 2: Number of PlonK Constraint for IEEE-754 Double-Precision GeLU, SeLU, and ELU: ZIP vs. Baseline (w/AG).

# Acts	GeLU			SeLU			ELU		
	ZIP	Baseline (w/AG)	|C| Gain†	ZIP	Baseline (w/AG)	|C| Gain†	ZIP	Baseline (w/AG)	|C| Gain†
2^0	1.7×10^4	1.7×10^7	**1004×**	1.2×10^4	5.3×10^6	**442×**	1.1×10^4	4.8×10^6	**452×**
2^4	2.3×10^5	2.8×10^8	**1227×**	1.5×10^5	8.0×10^7	**540×**	1.4×10^5	7.7×10^7	**565×**
2^8	3.2×10^6	4.4×10^9	**1353×**	2.1×10^6	1.3×10^9	**602×**	1.9×10^6	1.2×10^9	**636×**
2^{12}	4.8×10^7	6.2×10^{10}	**1293×**	3.0×10^7	2.0×10^{10}	**667×**	2.7×10^7	1.9×10^{10}	**707×**
2^{16}	7.6×10^8	9.2×10^{11}	**1212×**	4.7×10^8	2.4×10^{11}	**590×**	4.2×10^8	2.8×10^{11}	**664×**

† "|C| Gain" indicates how many times smaller ZIP circuit is compared to Baseline (w/AG).

$[-30, 30]$, $[-30, 30]$, and $[-5, 5]$, respectively. For ELU, we used $m = 7$ polynomial pieces of degree $k = 6$ (public table size $|\mathbf{c}| = 36$, $|\mathbf{c}'| = 7$, private vector size $|\mathbf{a}| = 6$, $|\mathbf{a}'| = 2$) with the approximation error $\delta = 9{\times}10^{-4}$. For SeLU, we employed $m = 6$ polynomial pieces of degree $k = 7$ ($|\mathbf{c}| = 35$, $|\mathbf{c}'| = 6$, $|\mathbf{a}| = 7$, $|\mathbf{a}'| = 2$) with $\delta = 9{\times}10^{-4}$. For GeLU we used $m = 8$ of degree $k = 10$ ($|\mathbf{c}| = 70$, $|\mathbf{c}'| = 8$, $|\mathbf{a}| = 10$, $|\mathbf{a}'| = 2$) with $\delta = 9 \times 10^{-2}$. Outside these intervals, we clamped inputs to the nearest polynomial piece [33]. We configured gnark to use the BN254 elliptic curve for all PlonK CP-SNARK proofs.

Counterpart Comparison and Evaluation Metrics. To our knowledge, no prior work has provided precise zero-knowledge proofs for IEEE-754–compliant non-linear activation functions within an AI inference pipeline. We therefore benchmarked ZIP against three baselines, evaluating both performance (proving and verification time) and precision of activation proofs for GeLU, SeLU, and ELU:

- **Baseline (w/AP)**: We replaced each non-linear activation with its piecewise-polynomial approximation (error bound δ) and executed the network end-to-end in IEEE-754 double-precision.
- **Baseline (w/FP)**: We retained actual non-linear activations but used fixed-point arithmetic throughout the network.
- **Baseline (w/AG)**: We treated ELU, SeLU, and GeLU as full IEEE-754 double-precision operations, hardcoded directly into the arithmetic circuits using state-of-the-art gadgets for square root [20], *tanh* [66], and IEEE-754 addition/multiplication [3].

We selected GeLU, SeLU, and ELU because they (1) are widely adopted in modern, state-of-the-art AI models and (2) incur significant arithmetic circuit overhead. We used the same PlonK CP-SNARK backend [25] for both ZIP and Baseline (w/AG).

5.3 Overall Results

Constraint Complexity. Table 1 reports the total number of R1CS constraints for proving IEEE-754 double-precision GeLU, SeLU, and ELU in ZIP and Baseline (w/AG) (i.e., directly hardcoded into the arithmetic circuit). Across all activation counts, ZIP achieves a 2–3 order of magnitude reduction in circuit complexity. Specifically, for proving a single GeLU activation, ZIP uses only 5.8×10^3 R1CS constraints compared to 6.8×10^6 in Baseline (w/AG)-a 1,172× decrease. Even at 2^{16} activations, ZIP maintains a 1,482× improvement, reducing from 4.5×10^{11} to 3.0×10^8 R1CS constraints. SeLU and ELU show similarly large gains, with ZIP requiring 545×–748× fewer R1CS constraints for SeLU and 569×–791× fewer for ELU.

As shown in Table 2, in the PlonK setting, ZIP achieves comparable gains. For a single GeLU activation, the number of PlonK constraints drops from 1.7×10^7 to 1.7×10^4, a 1,004× improvement over Baseline (w/AG), and remain over 1,212× more efficient at 2^{16} activations. SeLU and ELU likewise require 442×–664× fewer PlonK constraints across all scales. Additionally, each activation proof in ZIP leverages two multi-lookup arguments to amortize polynomial evaluations without significant overhead.

Figure 1 illustrates the per-activation R1CS and PlonK constraint savings in ZIP for IEEE-754 double-precision GeLU, SeLU, and ELU. Thanks to shared subexpressions, the cost to prove each activation steadily decreases as the number of invocations grows. Specifically, to prove a single GeLU activation ZIP requires 5,830 R1CS constraints, which decreases to 4,599 at 2^{14} activations, about 21% reduction. Similarly, SeLU falls from 3,670 to 2,649 (28%), and ELU from 3,341 to 2,389 (29%). Regarding PlonK constraints, GeLU decreases from 17,112 to 11,654 (32%), SeLU from 11,687 to 7,195 (38%), and ELU from 10,660 to 6,490 (39%).

Impact of Precision. We quantify precision loss when the true activation functions are replaced by their piecewise polynomial approximations or by fixed-point implementations, while keeping all other hyperparameters identical. We used the UTKFace dataset [75], which contains over 20,000 face images (200×200) labeled by age (0–116), gender, and ethnicity. These images were fed to an adapted CNN from [45] with approximately 250K parameters (5

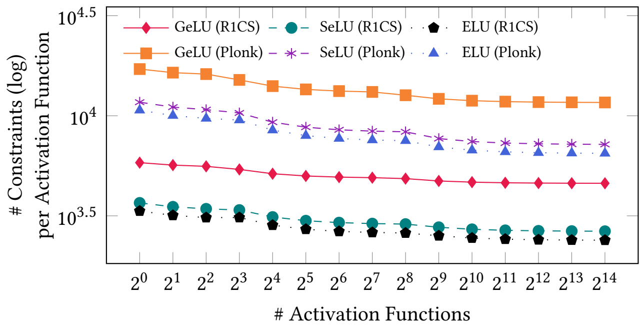

Figure 1: Per-activation constraint counts (R1CS and PlonK) for double-precision IEEE754–compliant GeLU, SeLU, and ELU in ZIP according to the number of activations.

convolutional, 5 pooling, and 3 fully connected layers). We used Mean Absolute Error (MAE) as the performance metric that indicates the average absolute difference between each predicted value and its true label over all test samples.

As shown in Table 3, our full-precision ZIP model–using IEEE-754 double-precision floating point and the native ELU activation–achieves an average test MAE of 6.15. Even with only seven activation layers and a small approximation error bound δ=0.0009, substituting the actual ELU with its piecewise polynomial approximation (i.e., Baseline (w/AP)) raises the MAE by 2–5%, which indicates a loss of precision. Moreover, Table 3 shows that quantizing all weights and inputs to fixed-point representation while retaining the actual ELU activation function (i.e., Baseline (w/FP)), the MAE increases significantly, demonstrating that fixed-point arithmetic severely degrades accuracy and prevents convergence.

Comparison with Prior Non-Linear Arguments under Fixed-Point Representation. ZIP employs lookup table arguments to prove non-linear activations under IEEE-754 floating-point semantics. While some prior works also employ lookup tables to support non-linear activations [30, 41, 57], they operate under fixed-point representation. Although targeting a different numerical format and not directly comparable, we provide a conservative comparison to highlight the distinctions and implications of our design.

The works in [30, 41, 57] introduce efficient ZKP techniques for non-linear activations by encoding model parameters and inputs as fixed-point values, trading precision for lower proving cost. For example, as shown in Table 4, proving one GeLU activation takes 240 ms in ZIP, compared to 0.38 ms in [30] and 0.025 ms in [41]. These fixed-point schemes incur quantization errors, whereas ZIP proves IEEE-754 double-precision computation of GeLU activations.

Beyond precision loss, non-linear arguments under fixed point representation may require large lookup tables for complex activations. For example, in [30], inputs are decomposed into small digits, and operations such as exponentiation, division, reciprocal, and square root are proved via multiple lookup tables. Although this method can be applied to many complex activations, it relies on fixed-point representation and requires constructing multiple lookup tables with 4,096 entries, with the proof overhead growing substantially as activations become more complex. Lu et al. [41] propose a VOLE-based ZKP framework that reduces bit-decomposition overhead by encoding non-linear layers as range and lookup proofs over exponent tables with several hundred entries. Sun et al. [57] present tlookup, a parallel lookup argument for non-arithmetic tensor operations (e.g., activations) that reduces each operation to lookup arguments over paired input/output tables. However, their approach requires large tables with 65,536 entries.

Table 3: Impact of Numerical Format on ELU Evaluation.

Configuration	Avg. Test MAE
ZIP (IEEE-754 double-precision)	6.16
Baseline (w/FP) (fixed-point)	no convergence

Table 4: ZIP vs. Prior Fixed-Point Methods for GeLU.

	$\mathcal{P}$(ms) / GeLU	LUT Entries†	Numeric Representation
ZIP	240	70	IEEE-754 double
Hao et al. [30]	0.38	4, 096	Fixed-point
Lu et al. [41]	0.025	∼ few ×100	Fixed-point
Sun et al. [57]	n/a	65, 536	Fixed-point

† "LUT" denotes the maximum lookup-table size used in the protocol.

5.4 Evaluation on Full AI Inference Pipelines

We evaluate the performance of our proposed techniques on complete AI inference pipelines. We selected LeNet-5 and mini-BERT models, which are mostly used for image classification and sentiment analysis, respectively. We also evaluate the impact of precision loss incurred by fixed-point arithmetic on mini-BERT.

Datasets and Model Parameters. For LeNet-5 [36], we used the MNIST dataset [17]. The model has roughly 60 K parameters (3 convolutional, 2 pooling, and 2 fully connected layers). We applied GeLU, SeLU, or ELU activations after every convolutional and fully connected layer. For mini-BERT [60], we used the SST-2 dataset [55], which comprises 67 K movie-review sentences labeled as positive or negative. The model has 4 layers, hidden size 256, containing around 11 M parameters. We represented all data in both datasets in the IEEE-754 double-precision format.

Experiments on LeNet-5. We measure end-to-end proving and verification time of ZIP on the LeNet-5 [36] using actual GeLU, SeLU, and ELU activation functions in IEEE-754 double precision.

As Table 5 shows, total proving times for the four non-linear layers are 19.78 min for GeLU, 15.02 min for SeLU, and 13.16 min for ELU, while verification remains stable at approximately 5.8 min across all three activation functions. To reduce circuit size for the linear layers, we use random linear combinations (RLC) from [24, 51] with a small relative tolerance to handle IEEE-754 rounding. Without RLC and thus preserving full numerical precision, end-to-end proving takes 40, 42, and 47 min for GeLU, SeLU, and ELU, respectively. In total, ZIP uses 32,451,489 R1CS constraints (resp. 66,133,656 PlonK constraints) for GeLU, 19,749,473 R1CS (resp. 54,765,423 PlonK) for SeLU, and 15,703,492 R1CS (resp. 42,887,728 PlonK) for ELU, and employs two multi-lookup arguments per activation with four pairing per lookup proof during verification.

Experiments on mini-BERT. We first evaluate the impact of numerical representation on the accuracy performance of mini-BERT [60] over the SST-2 dataset [55]. As shown in Table 6, the model (trained for 2 epochs on SST-2) achieves 85.65% accuracy under the IEEE-754 double precision. Representing all model parameters and

Table 5: Performance of ZIP on LeNet-5 using MNIST dataset.

	GeLU		SeLU		ELU	
	$\mathcal{P}$ (min)	$\mathcal{V}$ (min)	$\mathcal{P}$ (min)	$\mathcal{V}$ (min)	$\mathcal{P}$ (min)	$\mathcal{V}$ (min)
1st Act	13.89	4.21	11.19	4.19	9.30	4.23
2nd Act	5.23	1.43	3.49	1.43	3.42	1.43
3rd Act	0.37	0.11	0.25	0.11	0.24	0.11
4th Act	0.29	0.07	0.19	0.07	0.19	0.07
All linear layers †	0.89	0.07	0.89	0.07	0.89	0.07
Total	20.67	5.89	15.91	5.87	13.94	5.91

† All linear operations are proved via [20].

Table 6: Impact of Numerical Format on mini-BERT Accuracy.

Configuration	Avg. Accuracy (%)
ZIP (IEEE-754 double-precision)	85.65
Baseline (w/FP) (fixed-point)	77.14

Table 7: Performance of ZIP on mini-BERT using SST-2 Dataset.

Component	$\mathcal{P}$ (hr)	$\mathcal{V}$ (hr)
GeLU	2.44	0.73
Softmax †	0.09	0.03
LayerNorm †	0.003	0.001
All linear layers ‡	34.53	0.09
Total	37.06	0.85

† Times for Softmax and LayerNorm include only non-linear operations (exp and $1/\sqrt{x}$); their linear operations are counted under "linear layers".
‡ All linear operations are proved via [20].

inputs in fixed-point reduces accuracy to 77.14%. This indicates that fixed-point representation incurs precision loss in the mini-BERT model and negatively impacts its inference performance.

We then report the end-to-end proving and verification time of ZIP on mini-BERT inference with the SST-2 dataset, employing IEEE-754 double-precision implementations of GeLU, Softmax, and LayerNorm. Accordingly, we first interpolate piecewise-polynomial approximations of the non-linear operations $\exp(x)$ and $1/\sqrt{x}$, which appear in Softmax and LayerNorm. Each function is sampled at 1,000 points over the domains $[-8, 20]$ and $[0.001, 210]$, respectively. For $\exp(x)$ we use $m = 11$ polynomial pieces of degree $k = 7$ ($|\mathbf{c}| = 70$, $|\mathbf{c}'| = 11$, $|\mathbf{a}| = 7$, $|\mathbf{a}'| = 2$) with $\delta = 9 \times 10^{-3}$. For $1/\sqrt{x}$ we employ $m = 23$ polynomial pieces of degree $k = 4$ ($|\mathbf{c}| = 88$, $|\mathbf{c}'| = 23$, $|\mathbf{a}| = 4$, $|\mathbf{a}'| = 2$) with $\delta = 9 \times 10^{-3}$.

As shown in Table 7, using ZIP to prove non-linear layers under IEEE-754 semantics results in prover times of 2.44 hr for GeLU, 0.09 hr for Softmax, and 0.003 hr for LayerNorm. The corresponding verification times are 0.73 hr, 0.03 hr, and 0.001 hr, respectively. To complete the proof for the full inference pipeline, we directly apply the method from [20] to handle all remaining linear layers, including self-attention projections, output projections, feed-forward linears, and the classification head, which together require 37.06 hr to prove and 0.85 hr to verify.

Discussion. Although ZIP significantly reduces circuit size for non-linear activations compared to directly hardcoding them into circuits, its end-to-end delay for full AI inference remains high (Table 5, Table 7). We emphasize that our prototype is just a proof of concept to demonstrate correctness and feasibility, not performance optimization. Neither our implementation nor the ZKP libraries (i.e., gnark [6], Caulk [72]) were tuned for efficiency, and all reported results were obtained on a general-purpose CPU server without the hardware acceleration typically used in ML research. However, several approaches can be applied to ZIP to further reduce its latency and improve practicality, including distributed computation, hardware acceleration, and ZKP algorithmic improvements.

Specifically, we can consider distributed-prover architectures such as DIZK [69]. Because PlonK's prover and Caulk's preprocessing both rely on large Fast Fourier Transforms (FFTs) and Multi-Scalar Multiplications (MSMs), these compute-intensive kernels can be parallelized across many compute nodes instead of executing the entire zkSNARK prover on a single machine. This approach enables the prover to harness the combined compute power and memory of multiple machines, thereby overcoming single-node limitations. Since ZIP instantiates arithmetic circuits and lookup arguments using PlonK and Caulk, respectively, it inherits these benefits directly. Systems such as Pianist[38] further demonstrate how to restructure PlonK-style provers for highly parallel execution with minimal inter-node communication. By replacing univariate constraints with bivariate ones, sub-provers can process disjoint sub-circuits independently, while a master aggregator collects their constant-size commitments to form the final proof. We expect 100× latency reduction under these distributed architectures.

Another orthogonal approach to reduce latency is to explore hardware acceleration techniques. For instance, FFTs and MSMs parallelize well on GPUs, and prior work has demonstrated at least a 5× speedup in proving and a 2× speedup in verification [42, 43].

Beyond GPUs, FFT and MSM kernels can be mapped onto FPGA datapaths [8, 48] or implemented in custom ASICs [15, 16, 54]. This may lead to a reduction of proving time by two orders of magnitude (e.g., 600×) compared with our CPU baselines.

Taking these optimizations together, we expect ZIP to achieve cumulative improvements of several orders of magnitude, potentially reducing the concrete latency to the order of seconds, thereby making ZIP more practical. Such improvements open the door to supporting real-world AI inference with multi-million-parameter models (e.g., ResNet-50 [31], BERT-Base [18], GPT-2 [49], and RoBERTa-base [40]). Finally, our proposed relative-error method is generic and can be adopted with newer proof systems. As ZKP research advances with more efficient constructions (e.g., HyperPlonk [11] over PlonK), ZIP can potentially benefit from these algorithmic improvements, making the concrete delay more acceptable.

6 Related Work

Verifiable ML ensures ML computations are executed correctly, enabling clients to outsource training/inference to untrusted servers while still verifying correctness. The two main approaches are (1) verifiable computation (VC) and zero-knowledge proofs (ZKPs) over finite fields [7, 9, 13, 25, 28, 46, 63]; and (2) trusted execution environments (TEEs) [14], which run native floating-point instructions inside secure hardware enclaves. TEEs often run faster but require hardware trust and remain vulnerable to side-channel attacks.

Research in verifiable ML spans classic and deep models (e.g., decision trees, linear/SVMs, DNNs) [23, 29, 37, 39, 51, 56, 65–68, 73, 76], and falls into two broad categories. The first focuses on enhancing proof efficiency (e.g., circuit size, proving/verification time), particularly for non-linear computations, by approximating or precomputing activations in fixed-point representation or by employing large lookup tables [12, 30, 41, 57]. Approximation approaches replace activations with low-degree polynomials or piecewise-linear approximations [2, 26, 29, 76]. These methods reduce proving time and circuit size for non-linear computations but trade off numerical precision or introduce large table-storage requirements. The second targets proof precision for non-linear functions, also in fixed-point arithmetic. These systems construct linear building blocks (e.g., gadgets) for each activation [23, 37, 66], employ bit-decomposition [39, 66], or use interactive protocols that offload complex activations to the client [51] to avoid large circuits. Although they preserve higher-precision semantics for non-linear computations, these methods yield larger circuits, higher proving/communication costs, and often support only a narrow set of activations. Beyond inference/training, verifiable techniques has been applied to secure aggregation in federated learning [5, 70] and privacy-preserving inference [51, 68].

7 Conclusion

We presented ZIP, a zero-knowledge proof framework for AI inference with full IEEE-754 double-precision support. Our approach enables efficient, bounded-error proofs of non-linear activations and closes security gaps. Through targeted circuit optimizations, we shrink non-linear layer proofs by orders of magnitude, moving toward practical deployment of precise floating-point AI inference.

References

[1] 2019. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), 1–84. doi:10.1109/IEEESTD.2019.8766229

[2] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. 2024. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4316–4330.

[3] David W Archer, Shahla Atapoor, and Nigel P Smart. 2021. The cost of IEEE arithmetic in secure computation. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 431–452.

[4] MohammadHossein AskariHemmat, Sina Honari, Lucas Rouhier, Christian S Perone, Julien Cohen-Adad, Yvon Savaria, and Jean-Pierre David. 2019. U-net fixed-point quantization for medical image segmentation. In *International Workshop on Large-scale Annotation of Biomedical data and Expert Label Synthesis*. Springer, 115–124.

[5] Rouzbeh Behnia, Arman Riasi, Reza Ebrahimi, Sherman SM Chow, Balaji Padmanabhan, and Thang Hoang. 2024. Efficient secure aggregation for privacy-preserving federated machine learning. In *2024 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 778–793.

[6] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. 2025. *Consensys/gnark: v0.12.0*. doi:10.5281/zenodo.5819104

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

[8] Shahzad Ahmad Butt, Benjamin Reynolds, Veeraraghavan Ramamurthy, Xiao Xiao, Pohrong Chu, Setareh Sharifian, Sergey Gribok, and Bogdan Pasca. 2024. if-ZKP: Intel FPGA-Based Acceleration of Zero Knowledge Proofs. *arXiv preprint arXiv:2412.12481* (2024).

[9] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2075–2092.

[10] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2020. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium (USENIX Security 20)*. 1309–1326.

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

[12] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 560–574.

[13] Graham Cormode, Justin Thaler, and Ke Yi. 2011. Verifying computations with streaming interactive proofs. *arXiv preprint arXiv:1109.6882* (2011).

[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[15] Alhad Daftardar, Jianqiao Mo, Joey Ah-kiow, Benedikt Bünz, Ramesh Karri, Siddharth Garg, and Brandon Reagen. 2025. Need for zkSpeed: Accelerating HyperPlonk for Zero-Knowledge Proofs. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 1986–2001.

[16] Alhad Daftardar, Brandon Reagen, and Siddharth Garg. 2024. Szkp: A scalable accelerator architecture for zero-knowledge proofs. In *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*. 271–283.

[17] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[19] Paul Dierckx. 1995. *Curve and surface fitting with splines*. Oxford University Press.

[20] Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. 2024. Zero-Knowledge Location Privacy via Accurate Floating-Point SNARKs. *arXiv preprint arXiv:2404.14983* (2024).

[21] Xiaoyu Fan, Kun Chen, Guosai Wang, Mingchun Zhuang, Yi Li, and Wei Xu. 2022. Nfgen: Automatic non-linear function evaluation code generator for general-purpose mpc platforms. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 995–1008.

[22] Xiaoyu Fan, Kun Chen, Guosai Wang, Mingchun Zhuang, Yi Li, and Wei Xu. 2022. NFGen: Automatic Non-Linear Function Evaluation Code Generator for General-purpose MPC Platforms. https://github.com/Fannxy/NFGen.

[23] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. Zen: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *Cryptology ePrint Archive* (2021).

[24] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time. In *IFIP Congress*. 839–842.

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

[26] 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 *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1880–1894.

[27] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. 2022. Succinct zero knowledge for floating point computations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1203–1216.

[28] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 626–645.

[29] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *Advances in Neural Information Processing Systems* 30 (2017).

[30] 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 *33rd USENIX Security Symposium (USENIX Security 24)*. 3819–3836.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[32] William George Horner. 1819. XXI. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London* 109 (1819), 308–335.

[33] Mazharul Islam, Sunpreet S Arora, Rahul Chatterjee, Peter Rindal, and Maliheh Shirvanian. 2023. Compact: Approximating complex activation functions for secure computation. *arXiv preprint arXiv:2309.04664* (2023).

[34] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 512–527.

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

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. doi:10.1109/5.726791

[37] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2024. vcnn: Verifiable convolutional neural network based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 4254–4270.

[38] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. 2024. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1777–1793.

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

[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[41] Tao Lu, Haoyu Wang, Wenjie Qu, Zonghui Wang, Jinye He, Tianyang Tao, Wenzhi Chen, and Jiaheng Zhang. 2024. An efficient and extensible zero-knowledge proof framework for neural networks. *Cryptology ePrint Archive* (2024).

[42] Tao Lu, Chengkun Wei, Ruijing Yu, Chaochao Chen, Wenjing Fang, Lei Wang, Zeke Wang, and Wenzhi Chen. 2023. Cuzk: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023, 3 (2023), 194–220.

[43] Weiliang Ma, Qian Xiong, Xuanhua Shi, Xiaosong Ma, Hai Jin, Haozhao Kuang, Mingyu Gao, Ye Zhang, Haichen Shen, and Weifang Hu. 2023. Gzkp: A gpu accelerated zero-knowledge proof system. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 340–353.

[44] John C Mason and David C Handscomb. 2002. *Chebyshev polynomials*. Chapman and Hall/CRC.

[45] John McKinstry, Joshua Webb, Ganesh Janakiraman, and Benjamin Kelly. 2021. UTKFace Age Prediction – Basic Model. https://www.kaggle.com/code/johnmckinstry/utkface-age-prediction-gtech-final-project. Accessed: 2025-04-14.

[46] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* 59, 2 (2016), 103–112.

[47] Ivo Pereira, Ana Madureira, Nuno Bettencourt, Duarte Coelho, Miguel Ângelo Rebelo, Carolina Araújo, and Daniel Alves de Oliveira. 2024. A Machine Learning as a Service (MLaaS) Approach to Improve Marketing Success. In *Informatics*, Vol. 11. MDPI, 19.

[48] Xander Pottier, Thomas de Ruijter, Jonas Bertels, Wouter Legiest, Michiel Van Beirendonck, and Ingrid Verbauwhede. 2025. OPTIMSM: FPGA hardware accelerator for Zero-Knowledge MSM. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025, 2 (2025), 489–510.

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

[50] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. 2019. Efficiently stealing your machine learning models. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*. 198–210.

[51] Arman Riasi, Jorge Guajardo, and Thang Hoang. 2024. Privacy-Preserving Verifiable Neural Network Inference Service. In *2024 Annual Computer Security Applications Conference (ACSAC)*. 683–698. doi:10.1109/ACSAC63791.2024.00063

[52] Ahmed Salem, Michael Backes, and Yang Zhang. 2020. Don't trigger me! a triggerless backdoor attack against deep neural networks. *arXiv preprint arXiv:2010.03282* (2020).

[53] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. 2022. Dynamic backdoor attacks against machine learning models. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 703–718.

[54] Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. 2024. Accelerating zero-knowledge proofs through hardware-algorithm co-design. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 366–379.

[55] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. 1631–1642.

[56] Haochen Sun, Tonghe Bai, Jason Li, and Hongyang Zhang. 2024. Zkdl: Efficient zero-knowledge proofs of deep learning training. *IEEE Transactions on Information Forensics and Security* (2024).

[57] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4405–4419.

[58] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*. 601–618.

[59] Lloyd N Trefethen. 2008. IV. 21 Numerical Analysis. *The Princeton Companion to Mathematics (illustrated edition ed.). Princeton University Press, USA* (2008).

[60] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962* (2019).

[61] Dinusha Veluponnar, Lisanne L de Boer, Freija Geldof, Lynn-Jade S Jong, Marcos Da Silva Guimaraes, Marie-Jeanne TFD Vrancken Peeters, Frederieke van Duijnhoven, Theo Ruers, and Behdad Dashtbozorg. 2023. Toward intraoperative margin assessment using a deep learning-based approach for automatic tumor segmentation in breast lumpectomy ultrasound images. *Cancers* 15, 6 (2023), 1652.

[62] Yash Verma and Sunil Bharti. 2025. *AI in the Crosshairs: Understanding and Detecting Attacks on AWS AI Services with Trend Vision One*. https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/detecting-attacks-on-aws-ai-services-with-trend-vision-one Accessed 7 Jul. 2025.

[63] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. 2013. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 223–237.

[64] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing hyperparameters in machine learning. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 36–52.

[65] Haodi Wang, Rongfang Bie, and Thang Hoang. 2025. An Efficient and Zero-Knowledge Classical Machine Learning Inference Pipeline. *IEEE Transactions on Dependable and Secure Computing* 22, 2 (2025), 1347–1364. doi:10.1109/TDSC.2024.3435010

[66] Haodi Wang and Thang Hoang. 2023. ezDPS: An Efficient and Zero-Knowledge Machine Learning Inference Pipeline. *Proceedings on Privacy Enhancing Technologies* 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 *30th USENIX Security Symposium (USENIX Security 21)*. 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. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2218–2233.

[69] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. 2018. {DIZK}: A distributed zero knowledge proof system. In *27th USENIX Security Symposium (USENIX Security 18)*. 675–692.

[70] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. 2019. VerifyNet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security* 15 (2019), 911–926.

[71] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3121–3134.

[72] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup arguments in sublinear time. https://github.com/caulk-crypto/caulk.

[73] Hanlin Zhang, Peng Gao, Jia Yu, Jie Lin, and Neal N Xiong. 2021. Machine learning on cloud with blockchain: a secure, verifiable and fair approach to outsource the linear regression. *IEEE Transactions on Network Science and Engineering* 9, 6 (2021), 3956–3967.

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

[75] Zhifei Zhang, Yang Song, and Hairong Qi. 2017. Age progression/regression by conditional adversarial autoencoder. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5810–5818.

[76] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. 2021. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems* 32, 10 (2021), 2524–2540.