Activation-Efficient Neural Network Architectures

Dmytro Zakharov[1]

V.N. Karazin Kharkiv National University zakharov2025mp11@student.karazin.ua

Abstract. Due to the rise of neural network architectures, the need for their privacy and privacy-preserving computations became paramount. For that reason, the fields of zero-knowledge proofs and fully homomorphic encryption have been actively researched to work on top of neural networks. However, the activation functions used in neural networks are not always compatible with the existing arithmetizations of cryptographic primitives. In this paper, we analyze whether the activation functions used in neural networks can be optimized for the sake of making them more "cryptographically friendly".

Keywords: Neural Networks, Activation Functions, Architecture Optimizations, Zero-Knowledge, Fully Homomorphic Encryption

Contents

1 Introduction

Undeniably, the neural networks have become the cornerstone of modern machine learning and artificial intelligence. They are used in almost every field of computer science, from computer vision [KLG22, Hu23] to natural language processing [DAGY+25, VSP+23].

However, with the rise of neural networks, the rise of privacy concerns has followed. In particular, the following issues are of concern:

- How can we send the data x to the neural network f without revealing the data x itself? For example, can the person, having the medical image x, send it to the analyzing neural network f for the diagnosis without revealing x [XBF+14]? This example is illustrated in Figure 1, case **(a)**.

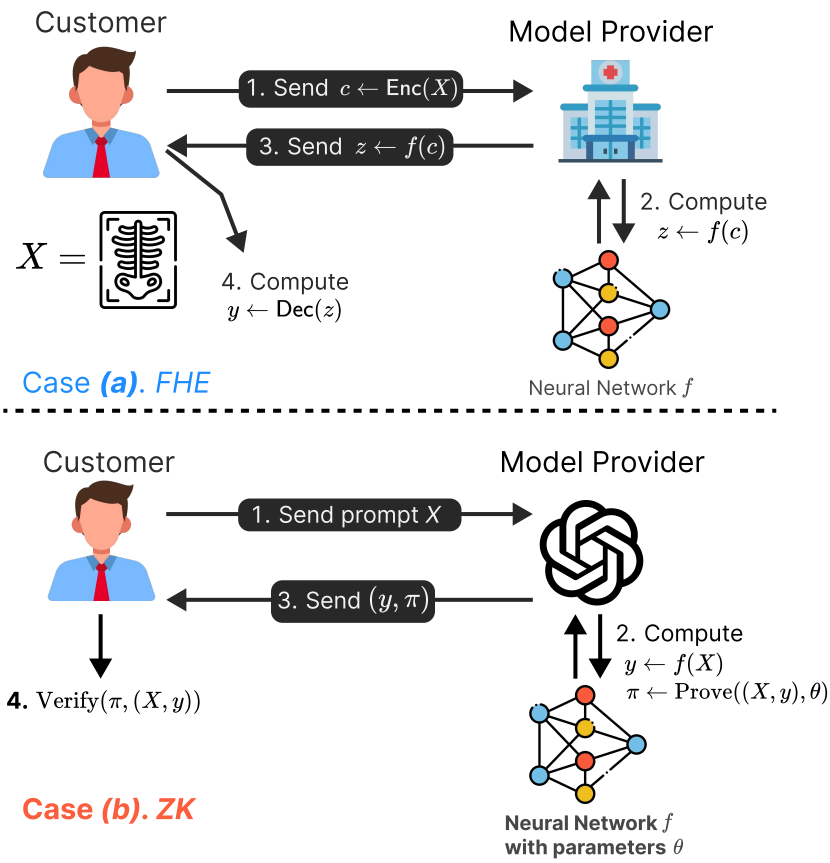

Figure 1: Privacy concerns in neural networks. Case (a) shows the privacy concern of sending the data x to the neural network f without revealing the data itself. The user encrypts the data x, sends to the neural network f, and receives the encrypted output, which can be decrypted by the user's key. Case (b) shows the use-case of zero-knowledge proofs for proving the neural network inference. The user sends the public input x to the private neural network f, and then gets the output y with a proof π that $y = f(x)$.

- How can we trust the output of the neural network? In other words, given input x and output y of the neural network f, how can we be sure that $y = f(x)$ [SCJ+24]? This example is similarly illustrated in Figure 1, case **(b)**.

- How can we prove that the neural network f was trained on the non-sensitive data [SBLZ23]? Can we train the neural network f on the encrypted dataset [CFR24]?

To address these issues, the fields of zero-knowledge proofs (ZKP) and fully homomorphic encryption (FHE) have been actively researched to work on top of neural networks, which we introduce in the Section 2. Despite the progress in the field, the biggest bottleneck is the activation functions used in neural networks. The activation functions are the non-linear transformations applied to the intermediate outputs of the neural network and are the core reason why the neural networks can approximate complex behaviours. Despite this, the activation functions complicate the arithmetization of the neural network inference for ZKP and FHE systems. We outline the core reasons in the Section 2.

1.1 Our Contribution

In this paper, we introduce the concept of *activation-efficient neural network architectures*. We analyze whether the activation functions used in neural networks can be optimized for the sake of making them more "cryptographically friendly". We propose a novel approach to designing neural network architectures that are more compatible with the existing arithmetizations of cryptographic primitives. We show that the proposed activation-efficient neural network architectures can be used to improve the efficiency of zero-knowledge proofs and fully homomorphic encryption systems working on top of neural networks.

2 Applications

2.1 Neural Networks

The neural network, vaguely speaking, is the complicated parameterized function $f(x;\theta)$, taking a high-dimensional input x and outputting the *prediction result* y using *weights* θ.

The most basic task in training neural networks is the so-called *supervised learning*. Suppose we are given the supervised dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in[N]}$ and we want to train the neural network f to have $f(\mathbf{x}_i) \approx \mathbf{y}_i$ (for example, $\mathbf{x}_i$ might be an image of either cat or dog while $y_i \in \{0,1\}$ is the image's label). We encapsulate our metric of model's "badness" in the loss function $\mathcal{L}$. For example, we might want to minimize the mean-squared error (MSE) on the given dataset:

$$\mathcal{L}(\mathcal{D};\theta) = \frac{1}{N}\sum_{i\in[N]} \|f(\mathbf{x}_i;\theta) - \mathbf{y}_i\|_2^2$$

Evidently, $\mathcal{L}$ is a function of parameters θ and the goal of the training process is to find the optimal $\theta^* = \arg\min_\theta \mathcal{L}(\mathcal{D};\theta)$ that minimizes the loss function. This is typically done by the iterative methods such as Adam [KB14] or SGD [Rud17]. So the typical optimization flow on the j^{th} step, if SGD optimizer is used, might be:

$$v_j \leftarrow \mu v_{j-1} + \eta\nabla_\theta\mathcal{L}(\mathcal{D};\theta), \quad \theta \leftarrow \theta - v_j,$$

where $\eta \ll 1$ is the learning rate and μ is the momentum (typically chosen between 0.9 and 1.0).

2.2 Zero-Knowledge Proofs

Suppose we have a statement $\mathbb{x}$ with the corresponding witness $\mathbb{w}$ (we typically write $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ for some relation $\mathcal{R}$). Typically, witness is some secret data that proves the statement $\mathbb{x}$ is true in polynomial or sub-polynomial time. For example, the statement $\mathbb{x}$ can be "I know the discrete logarithm of element $h := g^\alpha$ in the cyclic group $\mathbb{G} = \langle g \rangle$ of order q", and thus the witness $\mathbb{w}$ can be the value of $\alpha \in \mathbb{Z}_q$ while $\mathbb{x} = h \in \mathbb{G}$. The zero-knowledge proof systems have been extensively used in various applications, ranging from blockchain scalability solutions to privacy-preserving identity protocols [MKvWK19].

There are numerous types of zero-knowledge proof systems nowadays: Σ-proofs, zk-SNARKs [OREHS24], zk-STARKs [BSBHR18], Bulletproofs [BBB+18] etc, each balancing the trade-offs between the proof size, verification time, and proving time.

However, the majority of widely used zero-knowledge proofs are the so-called zk-SNARKs (zero-knowledge succinct non-interactive argument of knowledge). Let us briefly describe the zk-SNARKs.

Definition 1. A zk-SNARK is a zero-knowledge proof system protocol that allows one to prove the knowledge of a witness $\mathbb{w}$ for a statement $\mathbb{x}$ for relation $\mathcal{R}$ in zero-knowledge. The zk-SNARK consists of the following algorithms:

- $\mathrm{KeyGen}(1^\lambda) \to (\mathsf{pp}, \mathsf{vp})$: the key generation algorithm that generates the proving parameters pp and the verification parameters vp based on security parameter $\lambda \in \mathbb{N}$.
- $\mathrm{Prove}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) \to \pi$: the proving algorithm that generates the proof π for the statement $\mathbb{x}$ with the witness $\mathbb{w}$, based on the proving parameters pp.
- $\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) \to \{0, 1\}$: the verification algorithm that checks whether the proof π is valid for the statement $\mathbb{x}$ and outputs the bit 1 if the proof is valid and 0 otherwise.

Moreover, zk-SNARK satisfies the following properties:

- **Completeness:** for every statement $\mathbb{x}$ and the corresponding witness $\mathbb{w}$ (such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$), the proof π generated by the proving algorithm is valid:

$$\Pr\left[\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) = 1 \;\middle|\; \begin{array}{c} (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathrm{KeyGen}(1^\lambda) \\ \pi \leftarrow \mathrm{Prove}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) \\ (\mathbb{x}, \mathbb{w}) \in \mathcal{R} \end{array}\right] = 1$$

- **Soundness:** for every statement $\mathbb{x}$ and the corresponding witness $\mathbb{w}$ (such that $(\mathbb{x}, \mathbb{w}) \notin \mathcal{R}$), the proof π generated by the proving algorithm is invalid with overwhelming probability. Formally, for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\mathrm{Verify}(\mathsf{vp}, \pi, \mathbb{x}) = 1 \;\middle|\; \begin{array}{c} (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathrm{KeyGen}(1^\lambda) \\ (\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{vp}) \\ \pi \leftarrow \mathrm{Prove}(\mathsf{pp}, \mathbb{x}, \mathbb{w}) \end{array}\right] = \mu(\lambda),$$

  where μ is negligible in λ, that is $\lim_{\lambda \to \infty} \mu(\lambda)\lambda^\gamma = 0$ for all $\gamma \in \mathbb{R}$.

- **Argument of Knowledge:** for every statement $\mathbb{x}$, the proof π ensures the verifier that the prover *knows* the witness $\mathbb{w}$ for the statement $\mathbb{x}$.
- **Zero-Knowledge:** for every statement $\mathbb{x}$ and the corresponding witness $\mathbb{w}$, the proof π does not reveal any information about the witness $\mathbb{w}$.
- **Succinctness:** the proof π generated by the proving algorithm is of polylogarithmic size in the size of the computatinal problem N as well as the verification time T. In other words, typically we have $|\pi| = \mathcal{O}_\lambda(\log^\alpha N)$ and $T = \mathcal{O}_\lambda(\log^\beta N, |\mathbb{x}|)$.

For more information on zk-SNARKs, we refer the reader to [LHW+25].

However, to properly use zero-knowledge systems, one needs to somehow arithmetize the relation $\mathcal{R}$. Arithmetization always involves working over some finite field $\mathbb{F}$ (for concreteness, one might think of the prime field $\mathbb{F} = \mathbb{F}_p$ for some prime p). One of two most common ways to arithmetize the relations is to either use R1CS (Rank-1 Constraint Systems) or $\mathcal{P}$lon$\mathcal{K}$ish arithmetization. Regardless of the choice, the primary challenge of verifying whether $y = f(x)$ are the following:

- All operations must be conducted over the field $\mathbb{F}$.
- Each multiplication and addition operation adds a new constraint to the system, increasing dramatically the proving time and the proof size in certain proving systems. For example, as benchmarked in [Set19], the proving time of $2^{20} \approx 10^6$ constraints can take more than a minute in the Groth16 proving system [Gro16], roughly two minutes in Ligero [AHIV22], and up to eight minutes in Hyrax [WTas+17].
- Only addition and multiplication operations over two inputs are allowed.

Such arithmetization of f is called the **arithmetical circuit**[1], which we denote by C_f. Further, by A we cost of addition, by M we denote the cost of the multiplication operation. Finally, we specify the following theorem to understand the prover's complexity in terms of the number of additions and multiplications.

Theorem 1. *Let C_f be the arithmetical circuit of f. Let a be the number of additions and m be the number of multiplications, used in C_f: that is, the total cost of circuit evaluation is $a\mathsf{A} + m\mathsf{M}$. Then, the most common prover's time complexity in R1CS is $\mathcal{O}_\lambda(m \log m)$, while in $\mathcal{P}$lon$\mathcal{K}$ish arithmetization it is $\mathcal{O}_\lambda((m+a)\log(m+a))$.*

2.3 Homomorphic Encryption

One of the use-cases of activation efficient neural networks is the usage of the so-called *homomorphic encryption* (HE) schemes. Roughly speaking, such schemes allows one to encrypt two plaintexts (say, m_1 and m_2), encrypt them: $c_1 \leftarrow \mathsf{Enc}(m_1), c_2 \leftarrow \mathsf{Enc}(m_2)$, and then perform some operations on the ciphertexts (typically, additions and multiplications), which will result in the encryption $c = f(c_1, c_2)$ that, after decryption, will be equal to the result of the operation performed on the plaintexts: $f(m_1, m_2) = \mathsf{Dec}(c)$. Similarly to the zero-knowledge proofs, the homomorphic encryption schemes support only addition and multiplication operations over the certain finite structure.

Now, let us properly define the homomorphic encryption scheme.

Definition 2 (FHE Scheme)**.** Fully Homomorphic Encryption (FHE) scheme is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$:

- $\mathrm{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{ek})$: Given the security parameter $\lambda \in \mathbb{N}$, outputs the secret key sk kept secret by the user and public evaluation key ek.
- $\mathrm{Enc}(\mathsf{sk}, \mu) \to c$: Given the secret key sk and message μ, outputs the ciphertext c.
- $\mathrm{Dec}(\mathsf{sk}, c) \to \mu$: Given the secret key sk and ciphertext c, outputs the message μ.
- $\mathrm{Eval}(\mathsf{ek}, f, c_1, \dots, c_\ell) \to \widetilde{c}$: Given the evaluation key ek, function $f \in \mathcal{F}$ from the supported class of functions $\mathcal{F} \subseteq \{f : \mathbb{Z}_2^\ell \to \mathbb{F}\}$, and ciphertexts $c_1, \dots, c_\ell$, outputs the ciphertext of the result of the function f applied to the plaintexts corresponding to the ciphertexts.

[1] Formally, arithmetical circuit $C : \mathbb{F}^n \to \mathbb{F}^m$ is a directed acyclic graph with nodes labeled with $+$, $-$, or $\times$. Essentially, any arithmetical circuit defines an n-variate polynomial with an evaluation recipe.

We require the following properties:

- **Correctness:** for every supported circuit $f \in \mathcal{F}$ and inputs $\mu_1, \dots, \mu_\ell$, we have

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \widetilde{c}) = f(\mu_1, \dots, \mu_\ell) \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_j \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_j), j \in [\ell], \\ \widetilde{c} \leftarrow \mathsf{Eval}(\mathsf{ek}, f, c_1, \dots, c_\ell) \end{array}\right] \geq 1 - \mu(\lambda)$$

- **Semantic Security:** for any two messages μ_0, μ_1 it holds that

$$\left\{(\mathsf{ek}, c_0) : \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_0 \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_0) \end{array}\right\} \approx_C \left\{(\mathsf{ek}, c_1) : \begin{array}{c} (\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_0 \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_1) \end{array}\right\},$$

  where $\approx_C$ means the computational indistinguishability.

- **Compactness:** for every circuit $f \in \mathcal{F}$ and inputs $\mu_1, \dots, \mu_\ell$, let

$$\begin{aligned} &(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda), \\ &c_j \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_j),\; j \in [\ell], \\ &\widetilde{c} \leftarrow \mathsf{Eval}(\mathsf{ek}, f, c_1, \dots, c_\ell) \end{aligned}$$

  Then, sizes $\{|c_j|\}_{j \in [\ell]}$ and $|\widetilde{c}|$ depend only on the security parameter λ and independent of both the size $|f|$ and ℓ.

Now, let us consider the concrete fully homomorphic encryption scheme, namely the GSW scheme, introduced in [GSW13]. At a high level, the GSW utilizes the idea that the eigenvectors of matrices are preserved under the matrix addition and multiplication. Indeed, if $\mathbf{v}$ is the eigenvector of both C_1 and C_2 with eigenvalues λ_1 and λ_2, respectively, then $C_1\mathbf{v} = \lambda_1\mathbf{v}$, $C_2\mathbf{v} = \lambda_2\mathbf{v}$, and $(C_1 + C_2)\mathbf{v} = (\lambda_1 + \lambda_2)\mathbf{v}$, as well as $(C_1C_2)\mathbf{v} = \lambda_1\lambda_2\mathbf{v}$. That is, $\mathbf{v}$ is the eigenvector of $C_1 + C_2$ and C_1C_2 with eigenvalues $\lambda_1 + \lambda_2$ and $\lambda_1\lambda_2$, respectively. That being said, we will use eigenvector as the secret key, matrices as the ciphertexts, while the corresponding eigenvalues are the messages to be encrypted. Unfortunately, this way such scheme would not be secure as finding the eigenvectors is computationally easy. Therefore, we introduce the noise (error) term $\mathbf{e}$, changing the ciphertext/secret key relation to $C\mathbf{s} = \mu\mathbf{v} + \mathbf{e}$, where $\mu \in \mathbb{Z}_2$ is a message. Let us now give the formal definition of the GSW scheme.

Definition 3 (GSW Scheme.)**.** Fix parameters (m, n, q, χ) with sufficiently large m and some distribution χ. Let $\ell := (n+1)\log q$. Construction then works as follows:

- $\mathrm{KeyGen}(1^\lambda) \to \mathbf{s} \in \mathbb{Z}_q^{n+1}$. Sample a random vector $\mathbf{s}' \xleftarrow{R} \mathbb{Z}_q^n$ and output the vector $\mathbf{s} = (-\mathbf{s}', 1) \in \mathbb{Z}_q^{n+1}$ as the secret key.

- $\mathrm{Enc}(\mathbf{s} \in \mathbb{Z}_q^{n+1}, \mu \in \mathbb{Z}_2) \to C \in \mathbb{Z}_q^{\ell \times (n+1)}$. Sample the random matrix $A \xleftarrow{R} \mathbb{Z}_q^{\ell \times n}$ and error vector $\mathbf{e} \xleftarrow{R} \chi^\ell$. Build the matrix $B = (A, A\mathbf{s}' + \mathbf{e}) \in \mathbb{Z}_q^{\ell \times (n+1)}$ and output $C = B + \mu G$ for some fixed matrix G.

- $\mathrm{Dec}(\mathbf{s} \in \mathbb{Z}_q^{n+1}, C \in \mathbb{Z}_q^{\ell \times (n+1)}) \to \mu \in \mathbb{Z}_2$. Compute the vector $\mathbf{v} = C\mathbf{s}$. Output 0 if $\|\mathbf{v}\|_\infty < \delta q$ for small δ (that is, $\|\mathbf{v}\|_\infty$ is small, e.g., $\delta \approx \frac{1}{4}$), otherwise output 1.

Correctness. Let us check why such scheme is correct. Take the message $\mu \in \mathbb{Z}_2$ and encrypt it:

$$C \leftarrow \mathsf{Enc}(\mathbf{s}, \mu) = (A, A\mathbf{s}' + \mathbf{e}) + \mu G$$

Based on the LWE (Learning With Errors) assumption (more on that can be found in [ZTL22]), we have that $(A, A\mathbf{s}' + \mathbf{e}) \approx_C (A, \mathbf{u})$ for some random A and random $\mathbf{u} \in \mathbb{Z}_q^m$.

Therefore, the expression $(A, A\mathbf{s}' + \mathbf{e}) + \mu G$ is indistinguishable from random, thus the scheme is secure. Now, suppose we know the secret $\mathbf{s}$ and multiply it by the ciphertext:

$$\mathbf{v} = C\mathbf{s} = (A, A\mathbf{s}' + \mathbf{e}) \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} + \mu G \begin{pmatrix} -\mathbf{s}' \\ 1 \end{pmatrix} = \mu G\mathbf{s} + \mathbf{e}$$

If $\mu = 0$, then $\mathbf{v} = \mathbf{e}$, which is small. If $\mu = 1$, then $\mathbf{v} = \mathbf{e} + G\mathbf{s}$ and since $\mathbf{s}$ is arbitrary and G is fixed, the expression $G\mathbf{s}$ is indistinguishable from random, thus $\|\mathbf{v}\|_\infty$ should have a large norm (close to $\frac{1}{2}q \gg \delta q$).

Homomorphic Operations. To add two ciphertexts C_1 and C_2, we simply add them together: $C_1 \oplus C_2 \triangleq C_1 + C_2$. Indeed, if we are to decrypt the sum C, we would have:

$$C\mathbf{s} = C_1\mathbf{s} + C_2\mathbf{s} = (\mu_1 G\mathbf{s} + \mathbf{e}_1) + (\mu_2 G\mathbf{s} + \mathbf{e}_2) = \underbrace{(\mu_1 + \mu_2)}_{\text{new message}} G\mathbf{s} + \underbrace{(\mathbf{e}_1 + \mathbf{e}_2)}_{\text{new error}}$$

To define the product, we need an auxiliary function $\phi : \mathbb{Z}_q^{\ell \times (n+1)} \to \mathbb{Z}_q^{\ell \times (n+1) \log q}$ such that entries of $\phi(C)$ are all small (namely, they are in $\mathbb{Z}_2$) and the function ϕ is the inverse to the matrix G, that is $\phi(C)G = C$.

In this case, the product of two ciphertexts C_1 and C_2 is defined as $C_1 \otimes C_2 \triangleq \phi(C_1)C_2$. Decryption in such case is conducted as follows:

$$\begin{aligned}
(C_1 \otimes C_2)\mathbf{s} = \phi(C_1)(\mu_2 G\mathbf{s} + \mathbf{e}_2) &= \mu_2 \phi(C_1) G\mathbf{s} + \phi(C_1)\mathbf{e}_2 \\
&= \mu_2 C_1 \mathbf{s} + \phi(C_1)\mathbf{e}_2 \\
&= \mu_2(\mu_1 G\mathbf{s} + \mathbf{e}_1) + \phi(C_1)\mathbf{e}_2 \\
&= \underbrace{(\mu_1\mu_2)}_{\text{new message}} G\mathbf{s} + \underbrace{(\mu_2\mathbf{e}_1 + \phi(C_1)\mathbf{e}_2)}_{\text{new error}}
\end{aligned}$$

Error Propagation. The error propagation in the GSW scheme is the primary bottleneck which limits the number of operations that can be performed over the ciphertexts. To see why, consider the following proposition.

Proposition 1. *After each addition, the error in the GSW scheme is roughly doubled, while after each multiplication the error is multiplied by roughly* $n \log q$.

For that reason, we are oftentimes limited not only in terms of the execution time, but also in terms of the error that accumulates after each operation. For that reason, it is essential to reduce the error in the ciphertexts, which can be achieved partially by utilizing the activation-efficient neural networks.

3 Activation-Efficient Neural Networks

3.1 Arithmetical Circuits for Neural Networks

Now, the main question arises: can we theoretically design the arithmetical circuit (both for zero-knowledge proofs and homomorphic encryption) in such a way that it represents the neural network prediction with the appropriate accuracy? The answer is yes, which was specified in [XBF+14]. The primary theorem is the following.

Theorem 2. *Let f be a neural network with continuous non-linear transformations. Let $X \subset \mathbb{R}^n$ be compact. Then, for every $\varepsilon > 0$ there exists a polynomial p such that:*

$$\sup_{\mathbf{x} \in X} \|f(\mathbf{x}) - p(\mathbf{x})\| < \varepsilon$$

In other words, the neural network can be approximated by a polynomial with the arbitrary accuracy. Since any polynomial can be encoded as the arithmetical circuit, we can design the arithmetical circuit that represents the neural network prediction with the arbitrary accuracy. For that reason, we provide the following theorem, following the original study [XBF+14].

Theorem 3. *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be the encryption and decryption functions of a FHE system. Let* $f : \mathbb{R}^n \to \mathbb{R}^m$ *be the neural network consisting of continuous non-linear transformations. Let* $X \subset \mathbb{R}^n$ *be a compact. Suppose* sk *be the properly generated secret key. Then, for every* $\varepsilon > 0$ *there exists a circuit* $\widetilde{f}$ *such that*

$$\sup_{\mathbf{x} \in X} \| f(\mathbf{x}) - \mathsf{Dec}(\mathsf{sk}, \widetilde{f}(\mathsf{Enc}(\mathsf{sk}, \mathbf{x}))) \| < \varepsilon$$

Finally, we provide the theorem that connects the zero-knowledge proofs and the approximation of the neural networks.

Theorem 4. *Suppose* $f(\star; \boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$ *is the neural network, parameterized by weights* $\boldsymbol{\theta}$*. The prover, provided with the public input* $\mathbf{x}$*, wants to prove that* $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ *holds without revealing the parameters* $\boldsymbol{\theta}$*. Suppose* $\mathsf{Enc} : \mathbb{R}^n \to \mathbb{F}^n$ *encodes the real-valued vector to the vector of field elements while* $\mathsf{Dec} : \mathbb{F}^m \to \mathbb{R}^m$ *decodes it back. Then, for every* $\varepsilon > 0$ *there exists an arithmetical circuit* C_f *such that the proving system over the relation*

$$\mathcal{R} = \left\{ \begin{matrix} \mathbb{x} = (\mathbf{x}, \mathbf{y}, \varepsilon) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}_{>0} \\ \mathbb{w} = (\boldsymbol{\theta}) \end{matrix} \;\middle|\; \sup_{\mathbf{x} \in X} \| f(\mathbf{x}; \boldsymbol{\theta}) - \mathsf{Dec} \circ C_f \circ \mathsf{Enc}(\mathbf{x}) \| < \varepsilon \right\}$$

is complete and sound over the compact $X \subset \mathbb{R}^n$*.*

While the above two theorems provide the solid theoretical foundation, they do not help to design the circuits specifically and do not provide the computational efficiency of executing and using protocols over such circuits. Up until this point, the majority of existing approaches produce the cryptographic schemes that require the overwhelming resources to be used: see [CWSK24], for instance. The biggest bottleneck is the large matrix multiplications and non-linearities that are computationally expensive. Among these two issues, we try to mitigate the non-linearities by designing neural networks that use as few non-linearities as possible.

3.2 Activations in Neural Networks

Throughout the years, the neural network community has introduced a variety of activation functions which are used for different purposes. In this section, we provide the list of the most common activation functions and their compatibility with zero-knowledge proofs, homomorphic encryption, and optimization. The list is provided in Table 1.

Table 1: The most common activation functions and their applications.

Name	Formula	Where used?
ReLU	$\max\{0, x\}$	Cost-effective non-linearity
LeakyReLU$_\alpha$	$\max\{\alpha x, x\}$	Cost-effective non-linearity
Sigmoid $\sigma(x)$	$\frac{1}{1+e^{-x}}$	Binary classification
Tanh	$\frac{1-e^{-2x}}{1+e^{-2x}}$	Binary classification
Softmax	$\left\{e^{x_i} / \sum_{j \in [n]} e^{x_j}\right\}_{i \in [n]}$	Multi-class classification
Swish	$x\sigma(x)$	General-purpose non-linearity

Now, suppose given the activation function $g : \mathbb{R} \to \mathbb{R}$, our problem is to design the arithmetical circuit that represents the function g. One of the widely used methods is to approximate the function g by the polynomial $\widetilde{g}$ of the degree d. For example, this can be done using the Taylor series expansion:

$$\widetilde{g}(x) = \sum_{k=0}^{d} c_k x^k + \mathcal{O}(x^{d+1}), \quad c_k = \frac{g^{(k)}(0)}{k!}$$

Expansions of some of the most common activation functions are provided in Table 2. For example, the ReLU function can be approximated by the polynomial $0.47+0.5x+0.09x^2$ or, if desirable, with even higher-order degrees. Then, to use the activation function in the arithmetical circuit, we encode the coefficients $\{c_j\}_{j\in[d]}$ as the field elements and perform the polynomial evaluation over the field (or ring in FHE).

However, the polynomial approximation proved itself to be inefficient due to the significant error induced during the whole neural network execution. This has a rigorous reasoning, included in [LLPS93], which claims that every non-linear functions, **except for polynomials**, can approximate any continuous function on the compact.

Table 2: The most common activation functions and their applications.

Name	Approximation
ReLU	$0.47 + 0.5x + 0.09x^2$
Tanh	$0.51x - 0.04x^3 + 0.0011x^5$
Swish	$0.24 + 0.5x + 0.1x^2$

This motivates us to use the activation functions that are not polynomials, yet can be theoretically implemented in the arithmetical circuits. Obviously, all candidates involving the exponential or trigonometric functions are not suitable for the cryptographic protocols. For that reason, we consider the ReLU function and its modifications as the target activation functions. Our main claim is the following.

Proposition 2. *Implementing the* ReLU *function in the arithmetical circuit requires roughly* $\log_2|\mathbb{F}|$ *multiplication and* $2\log_2|\mathbb{F}|$ *addition gates where* $\mathbb{F}$ *is the underlying field.*

Proof. The ReLU function is defined as $\mathsf{ReLU}(x) = \max\{0, x\}$. To implement it in the arithmetical circuit, we need to check the sign of x. For that reason, we need to decompose the number x into the binary representation: $x = \sum_{i=0}^{\ell-1} x_j 2^j$ where $\ell = \lceil \log_2|\mathbb{F}| \rceil$. Since each x_j must be either 0 or 1, we impose the constraint $x_j(x_j-1) = 0$, costing us one multiplication and one addition gate. Then, to sum all the x_j's, we need additional $\log_2|\mathbb{F}|$ addition gates. The total cost is thus $\log_2|\mathbb{F}| \cdot \mathsf{M} + 2\log_2|\mathbb{F}| \cdot \mathsf{A}$. Finally, if x_s is the sign bit (for some fixed $s \in [\ell]$), it suffices to output $x_s \cdot x$: if $x_s = 0$, then the output is 0, otherwise the output is x. This requires one additional multiplication gate. ■

This way, we can implement any ReLU-like functions in the arithmetical circuits without losing the accuracy. Yet, what other functions are we left with? MobileNet papers [SHZ+18, HSC+19] introduced the optimized versions for sigmoid and swish functions, called *hard sigmoid* and *hard swish*. To define them, consider the ReLU6 function:

$$\text{ReLU6}(x) = \min\{\max\{0, x\}, 6\}$$

Then, to define the hard sigmoid and hard swish functions, we use the following formulas:

$$\text{h-sigmoid}(x) = \frac{1}{6}\text{ReLU6}(x-3)$$
$$\text{h-swish}(x) = x \cdot \text{h-sigmoid}(x)$$

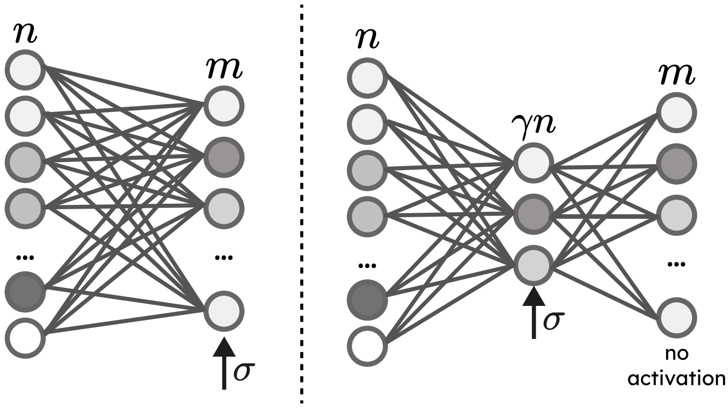

Figure 2: The Encoder-Decoder layer.

Note that such functions cost roughly twice as much as the ReLU function, since we need to perform two comparisons instead of one. However, they are still efficient enough to be used in the cryptographic protocols.

Finally, the Softmax layer can be implemented as the regular maximum function: instead of computing the vector $\mathrm{Softmax}(\mathbf{x})$, we simply output the one-hot vector $\mathbf{y}$ where $\mathbf{y}_i = \mathbf{1}[x_i = \max_j x_j]$ with $\mathbf{1}[\cdot]$ being the indicator function.

3.3 Optimizing the Number of Activations

Another problem is the number of activations. Consider the simplest construction: the neural network with the single layer. Dropping the bias term, the forward pass is defined as $f(\mathbf{x}; W) = \sigma(W\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^n$, $W \in \mathbb{R}^{m \times n}$, and σ is the activation function (think of ReLU). Now suppose we want to implement the forward pass in the arithmetical circuit with the arbitrary field $\mathbb{F}$, where its bitsize is $\ell := \lceil \log_2 |\mathbb{F}| \rceil$. What is the complexity of such approach?

Proposition 3. *The complexity of the forward pass in the neural network with the single layer is $\mathcal{O}(m(n+\ell))$ multiplications and additions.*

Proof. Note that the matrix-vector multiplication $W\mathbf{x}$ costs $\mathcal{O}(mn)$ multiplications and additions. Then, the activation function σ costs $\mathcal{O}(\ell)$ multiplications and additions. Since the activation function is applied to each element of the vector, the total cost is $\mathcal{O}(m(n+\ell))$. ■

Is there any way to reduce the number of activations? The answer is yes. The core idea that we propose is to use the Encoder-Decoder architecture, where the activation function is applied only to the output of the Encoder. Let us define the architecture.

Definition 4. The **Encoder-Decoder layer** between n and m neurons in input and output, respectively, with h hidden units, is defined as follows: let $E \in \mathbb{F}^{n \times h}$ be the encoder matrix that squeezes the input to the size of h. Let $D \in \mathbb{F}^{h \times m}$ be the decoder matrix that expands the output back to the original output size m. The forward pass is defined as

$$f(\mathbf{x}; D, E) = D\sigma(E\mathbf{x})$$

The Encoder-Decoder layer is illustrated in Figure 2.

Why does it help? Consider the following proposition.

Proposition 4. *The complexity of the forward pass in the neural network with the Encoder-Decoder layer is $\mathcal{O}(h(n+m+\ell))$ multiplications and additions.*

Proof. The first step is to compute the matrix-vector multiplication $E\mathbf{x}$, which costs $\mathcal{O}(nh)$ multiplications and additions. Then, the activation function σ is applied to the output of the Encoder, which costs $\mathcal{O}(h\ell)$ multiplications and additions. Finally, the matrix-vector multiplication $D(E\mathbf{x})$ costs $\mathcal{O}(\gamma mn)$ multiplications and additions. The total cost is $\mathcal{O}(nh + h\ell + hm)$. ■

It is not immediately clear why the Encoder-Decoder architecture is beneficial. Let us consider the concrete example.

Example 1. Suppose $n = 1000$, $m = 10000$, and $h = 10$. Suppose the field bitsize is $\ell = 254$ (which is the case for the majority of zk-SNARKs protocols such as Groth16). Then, the complexity of the forward pass in the neural network with the single layer is $10000(1000 + 254) \approx 10^7$ multiplications and additions. In turn, the complexity of the forward pass in the neural network with the Encoder-Decoder layer is $10(1000 + 10000 + 254) \approx 10^5$ multiplications and additions. The reduction is 100 times, which is significant!

Intuition. Without accounting for the activations, the boost in terms of number of operations is $\frac{mn}{h(m+n)}$. In particular, for $n = m$, the boost is $n/2h$, while for $m \gg n$, the boost is roughly n/h. Since we typically want to have $h \ll n$, this results in the significant boost. This is the reason why the Encoder-Decoder architecture is beneficial for the large neural networks.

However, due to the fact that we have much less activations in the Encoder-Decoder architecture, we expect the accuracy to be lower. But how much lower? This is the primary goal of our next section where we conduct the experiments to assess the accuracy of the Encoder-Decoder architecture under various "squeezing" factors. Formally, we define the following factor of squeezing.

Definition 5. Activation Squeezing Factor γ is defined as the ratio of activations in the Encoder-Decoder architecture to the number of activations in the neural network with the single regular layer. Following the notation in this section, we have $\gamma = h/m$.

4 Experiment

We consider the Fashion-MNIST dataset [XRV17], which consists of 60000 RGB images of size 28×28 pixels. Similarly to the classical MNIST, this dataset consists of 10 classes. Example images from the dataset can be seen in Figure 3. The primary reason why we employ this dataset instead of the classical MNIST is that the latter is too simple and thus the results are much less interesting.

We consider the neural network with the Encoder-Decoder layer with $28 \cdot 28 = 784$ neurons in the input, connected with the Encoder-Decoder layer with varying parameter h (number of hidden neurons). Namely, the Python (TensorFlow [AAB+15]) code to construct the Encoder-Decoder layer is provided below:

```python
def ed_model(hidden_units: int) -> tf.keras.models.Model:
    return tf.keras.models.Sequential([
        tf.keras.layers.Input(shape=(28,28,1)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(hidden_units, activation=None),
        tf.keras.layers.ReLU(),
        tf.keras.layers.Dense(100, activation=None),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

Figure 3: Example images from the Fashion-MNIST dataset [XRV17].

Then, we train the model for different values of h and evaluate the maximum achieved accuracy on the test set. We use the SGD optimizer with the learning rate 10^{-3} and momentum 0.9. For each h, we use the batch size of 128 and train the model for 100 epochs. The results are provided in Figure 4.

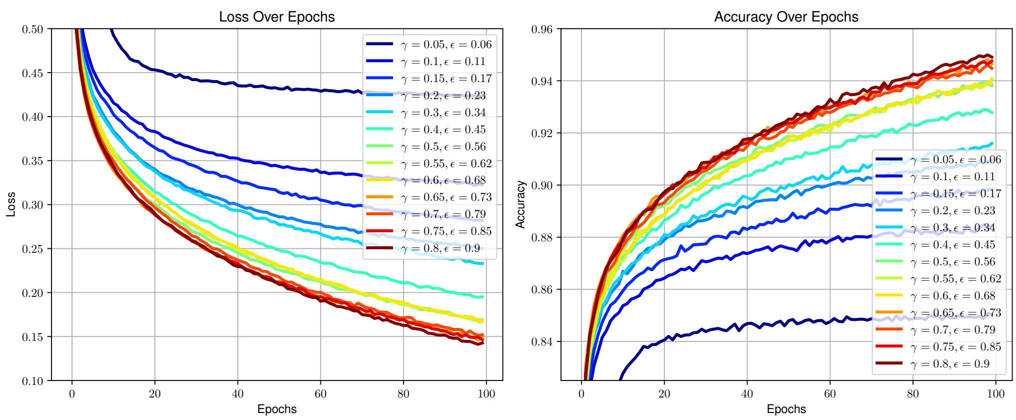

Figure 4: Accuracy and loss over epochs for different values of h. By ϵ we denote the ratio of new number of parameters in the model to the original number of parameters.

As can be seen, as expected, the decrease in the number of activations (parameter h) results in worse accuracy: in fact, the accuracy drops by 10% for $\gamma = 0.05$ compared to the best recorded accuracy (which is 95%). However, very solid results can still be obtained

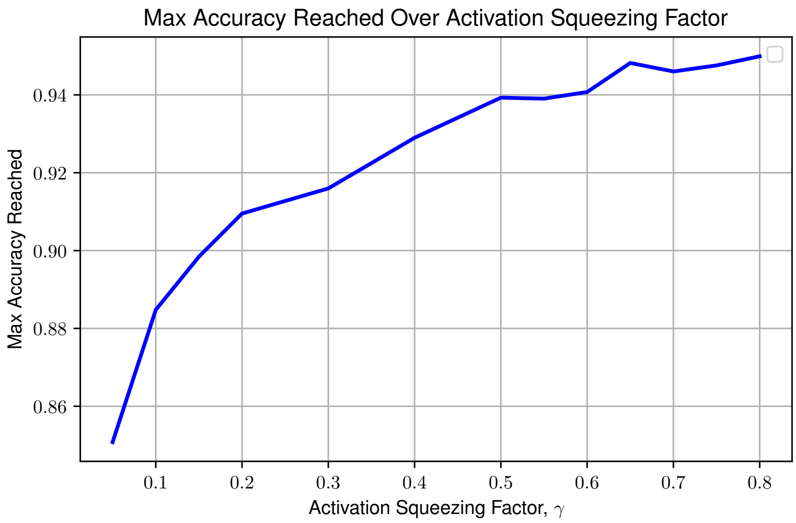

Figure 5: The dependence of max accuracy on the parameter γ.

using relatively small values of γ: namely, for $\gamma = 0.3$ the accuracy is close to 92%. We additionally depict the dependence of max accuracy on the parameter γ, which is specified in Figure 5.

This way, we can indeed reduce the number of activations in the neural network, subsequently reducing the number of operations in the arithmetical circuit by sacrifising a non-significant amount of accuracy. This is the main result of our paper.

4.1 Future Work

Note that we presented the substitution only for the fully-connected layers. The next step is to extend the results to the convolutional layers. One of the examples of such activation-friendly convolutional layers is the so-called Squeeze-and-Excitation layer, presented by [HSA+19], see Figure 6 for illustration. Given input volume $\mathbf{X} \in \mathbb{R}^{W \times H \times C}$ (where C is the number of channels and $W \times H$ is the image size), we first apply the Global Average Pooling to extract the channel-wise average values $\mathbf{z} \in \mathbb{R}^C$. Then, $\mathbf{z}$ is passed through the Encoder-Decoder Layer with the scaling vector $\mathbf{s} \in \mathbb{R}^C$ as the output. Finally, each image $\mathbf{X}_{[:,:,i]}$ gets multiplied by the corresponding scaling factor s_i for $i \in [C]$. The main advantage of this layer is that it is very lightweight and can be used in the arithmetical circuits since all the non-linear operations are conducted in the Encoder-Decoder layer, which is only C neurons wide.

The natural question arising is whether we can figure out more activation-friendly layers in the context of Convolutional Neural Networks or, possibly, in Recurrent Neural Networks and Large Language Models.

Finally, more experiments are needed to fully understand the balance between the accuracy and the number of activations. In particular, we would like to understand the trade-off between the number of activations and the accuracy for the different datasets.

5 Conclusion

In this paper, we introduced the notion of activation-friendly neural networks and explained the wide range of applications where they can be used. We proceeded to show that the Encoder-Decoder architecture can be used to reduce the number of activations in the

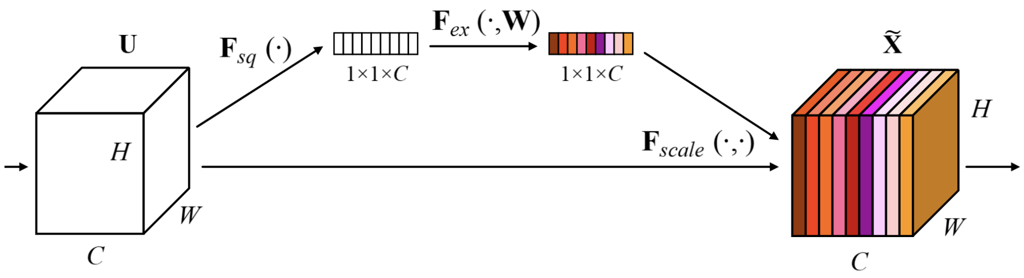

Figure 6: The Squeeze-and-Excitation layer. Figure taken from the original paper [HSA+19].

neural network, while keeping the accuracy relatively high. We conducted the experiments on the Fashion-MNIST dataset and showed that the accuracy drops by only 2% when both the number of activations and parameters is reduced by more than 3 times. We also provided the theoretical foundation for the Encoder-Decoder architecture and showed that it is indeed beneficial in terms of the number of operations in the arithmetical circuit. Finally, we discussed the future work and the possible extensions of the results to the convolutional layers and other types of neural networks.

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, and Craig Citro et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AHIV22] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. Cryptology ePrint Archive, Paper 2022/1608, 2022.

[BBB+18] Benedikt Bunz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. pages 315–334, 05 2018.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[CFR24] Luca Colombo, Alessandro Falcetta, and Manuel Roveri. Training encrypted neural networks on encrypted data with fully homomorphic encryption. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '24, page 64–75, New York, NY, USA, 2024. Association for Computing Machinery.

[CWSK24] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 560–574, New York, NY, USA, 2024. Association for Computing Machinery.

[DAGY+25] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, and et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[HSA+19] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019.

[HSC+19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[Hu23] Anthony Hu. Neural world models for computer vision, 2023.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KLG22] Maciej Krzywda, Szymon Lukasik, and Amir H. Gandomi. Graph neural networks in computer vision - architectures, datasets and common approaches. In *2022 International Joint Conference on Neural Networks (IJCNN)*, page 1–10. IEEE, July 2022.

[LHW+25] Junkai Liang, Daqi Hu, Pengfei Wu, Yunbo Yang, Qingni Shen, and Zhonghai Wu. SoK: Understanding zk-SNARKs: The gap between research and practice. Cryptology ePrint Archive, Paper 2025/172, 2025.

[LLPS93] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[MKvWK19] Eduardo Morais, Tommy Koens, Cees van Wijk, and Aleksei Koren. A survey on zero knowledge range proofs and applications. *SN Applied Sciences*, 1(8):946, 2019.

[OREHS24] Bjorn Oude Roelink, Mohammed El-Hajj, and Dipti Sarmah. Systematic review: Comparing zk-snark, zk-stark, and bulletproof protocols for privacy-preserving authentication. *SECURITY AND PRIVACY*, 7(5):e401, 2024.

[Rud17] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[SBLZ23] Haochen Sun, Tonghe Bai, Jason Li, and Hongyang Zhang. zkDL: Efficient zero-knowledge proofs of deep learning training. Cryptology ePrint Archive, Paper 2023/1174, 2023.

[SCJ+24] Tobin South, Alexander Camuto, Shrey Jain, Shayla Nguyen, Robert Mahari, Christian Paquin, Jason Morton, and Alex 'Sandy' Pentland. Verifiable evaluations of machine learning models using zksnarks, 2024.

[Set19] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550, 2019.

[SHZ+18] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[WTas+17] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2017/1132, 2017.

[XBF+14] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data, 2014.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[ZTL22] Zhiyong Zheng, Kun Tian, and Fengxia Liu. *Learning with Error*, pages 53–98. 12 2022.