PROVABLE WATERMARK EXTRACTION

Suyash Bagad, Yuval Domb, Ashutosh Marwah, Omer Shlomovits & Tomer Solberg
Ingonyama
{suyash,yuval,ash,omer,tomer}@ingonyama.com

ABSTRACT

Introducing zkDL++, a novel framework designed for provable AI. Leveraging zkDL++, we address a key challenge in generative AI watermarking—maintaining privacy while ensuring provability. By enhancing the watermarking system developed by Meta, zkDL++ solves the problem of needing to keep watermark extractors private to avoid attacks, offering a more secure solution. Beyond watermarking, zkDL++ proves the integrity of any deep neural network (DNN) with high efficiency. In this post, we outline our approach, evaluate its performance, and propose avenues for further optimization.

1 INTRODUCTION

Stable Signature (Fernandez et al., 2023) is a watermarking (WM) scheme introduced by Meta AI to allow for identification of pictures generated by their latent diffusion model (LDM). It uses a convolution neural network (CNN) to extract the WM, and has the advantage of being very robust to image tampering, allowing for detection even if the image was filtered, cropped, compressed, etc. However, Stable Signature suffers from the drawback of having to keep the extractor model private, as exposing it will make the WM susceptible to attacks. This drawback can prove meaningful if at any point there is a controversy about ownership of a certain image. While Meta, or any other company using such a WM method, can run the extractor and see that a certain image was generated by their model, they cannot prove it to an external arbiter, unless they expose the weights of the extractor model. Such exposure may lead to obsolescence of the WM system.

We propose using the emerging technology of Zero-Knowledge Proof (ZKP) as a solution to this problem. In a ZKP scheme, a prover wishes to convince a verifier that they have knowledge of a certain witness W, which obeys a certain relation $F(X, W) = 0$, without revealing anything about what W is. The relation F is generally represented by an arithmetic circuit and is public. The variable X contains public input and output to the circuit. In the case of stable signature, the relation F is the architecture of the WM extractor, with its output compared to the key, and it is satisfied if and only if its boolean output matches the existence of the WM in the input image. The public IO $X = (x_{in}, x_{out})$ includes the image and the boolean output, and the private witness $W = (w, k)$ includes the weights of the extractor and the key (figure 1). A Verifier can then run an algorithm that tests the proof and returns True if the relation is indeed satisfied, and False otherwise.

An added complexity here is the need to prove that the weights have not been chosen to get the desired result for the particular image. This is easily fixed:

1. Compute a cryptographic commitment to the weights $C = c(W)$. This can be thought of as the hash of W. The commitment is published together with the WM system.
2. Add C to X in F. This is the representation of W that is included in the proof, so the proof is only valid if indeed $C = c(W)$.

1.1 REAL-WORLD APPLICATION OF PROVABLE WATERMARKING EXTRACTION

According to Meta, the end goal is to be able to label AI generated content on social platforms. Technologies like Stable Signature can significantly enhance detection accuracy by embedding robust watermarks into AI-generated media. Meta can add these watermarks to content generated by their models, but users cannot verify them independently because the extractor model's weights must

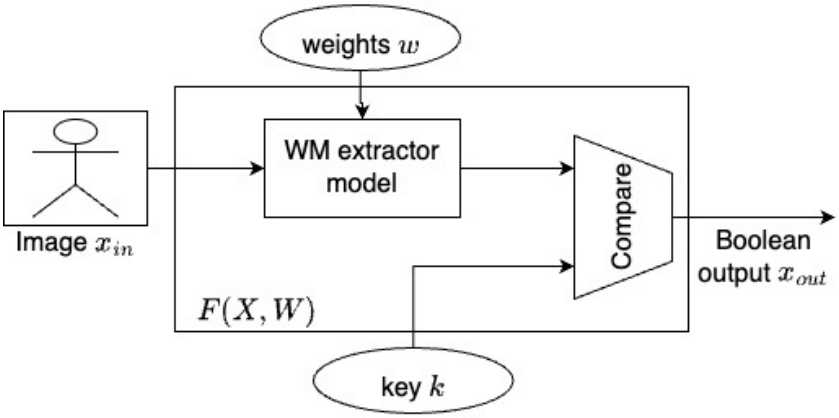

Figure 1: The WM extraction relation

remain private. Moreover, this challenge is compounded by the fact that each company, like Google, has its own proprietary watermarking system that is incompatible with others. ZKPs complement perfectly the shortcomings of such a system. For AI-generated images (the same approach applies to audio as well, See Roman et al. (2024)), Meta can attach a proof of watermark extraction, akin to a digital signature, without revealing sensitive model details. Clients can then run ZK verification for each image in their feed, which enables automatic labeling of AI-generated content. Additionally, other companies like Google can integrate their own watermark extraction proofs, allowing the system to distinguish between different AI models. Meta will handle the resource-intensive task of running the prover, while clients—such as mobile devices—can handle the lightweight verification. Since the watermark is robust against transformations (like cropping or compression), Meta can rerun the prover to account for any transformations and produce new proofs, as demonstrated with approaches like VIMz (Dziembowski et al., 2024).

2 Integrity Proofs for DNNs

In Meta's Stable Signature, the watermark extractor is a CNN specifically designed to analyze watermarked images. Let's focus on images sized 128×128 pixels. The figure 2 shows the architecture of Meta's extractor DNN, which consists of 9 convolution layers, an average pooling layer, and a fully connected layer. This model contains approximately 38 million parameters.

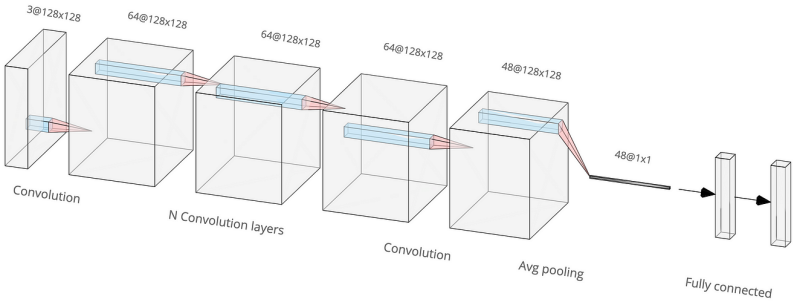

Figure 2: Stable Signature's DNN architecture

Current industry solutions for proving DNN inference often rely on zkVMs. However, zkVMs are not application-specific and introduce unnecessary complexity, making them less suitable for this task. In contrast, two recent academic works, zkCNN (Liu et al., 2021) and zkDL (Sun et al., 2023),

propose SNARK frameworks tailored to proving DNN inference, both based on the Sumcheck protocol (Lund et al., 1992).

As shown in figure 3, zkCNN leverages the GKR protocol (Goldwasser et al., 2015), while zkDL independently proves each layer of the DNN. The trade-offs between the two are notable:

- zkCNN commits only to the inputs and outputs of the network, whereas zkDL requires commitment to all intermediate inter-layer values.
- However, zkDL allows for concurrent processing of layer proofs, while zkCNN must process them sequentially.
- Another advantage of zkDL is its flexibility — each layer-proof can use a different proving system, as long as shared commitments are maintained.

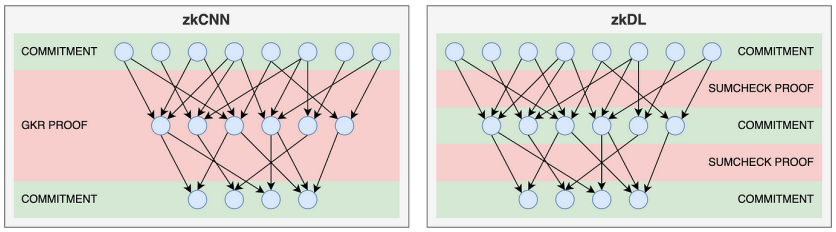

Figure 3: zkCNN vs. zkDL

In section 3, we outline how we integrated concepts from both zkDL and zkCNN to develop a proof-of-concept for a DNN-SNARK framework, which we call zkDL++.

Before diving into our implementation, let's first compare the proof construction run-times for a network of two fully-connected layers. This analysis evaluates zkDL/zkDL++, Jolt zkVM (Arun et al., 2023), and SP1 zkVM (Labs, 2024). As illustrated in the diagram below, the input to the network of size L is reduced to size $L/4$ in the first fully-connected layer, which then passes through a second fully-connected layer that produces an output of size $L/8$. Both inputs and network parameters are randomized uniformly.

We start by implementing the model in all three frameworks. We sample the input vector as a random byte-vector and arithmetize the fully-connected layers computation (i.e. matrix-vector products) in the three frameworks. This is followed by proof generation and benchmarking. The code for our evaluation is available at zkDL++ (available soon), Jolt, and SP1.

The comparison shows that zkDL/zkDL++ outperforms both Jolt and SP1 in prover run-time performance, especially as input sizes increase. While Jolt and SP1 suffer significant performance degradation with larger inputs, zkDL/zkDL++ maintains relatively low run-times, making it ideal for systems that demand high performance and scalability. For example, when the input size is 2^{10}, zkDL generates a proof around 200 times faster than SP1, as is presented in the prover run-time graph in figure 4.

In this scenario, the number of model parameters is 0.3 million — still much lower than a typical DNN model. In such cases, zkDL++ would likely be the only framework capable of practically generating proofs of integrity.

It is worth noting all the experiments were performed on the same hardware configuration, which consists of a powerful CPU (13th Gen Intel Core i9–13900K) and a high-end GPU (NVIDIA GeForce RTX 4080). Although the machine is equipped with a capable GPU, the experiments were not specifically optimized to fully utilize the GPU's potential. GPU optimization could potentially yield faster results, but that wasn't the focus of our experiments.

3 zkDL++

zkDL++ can be viewed as a hybrid of zkDL and zkCNN, designed to optimize both approaches. While zkDL and zkCNN represent two extremes — proving each layer independently versus proving

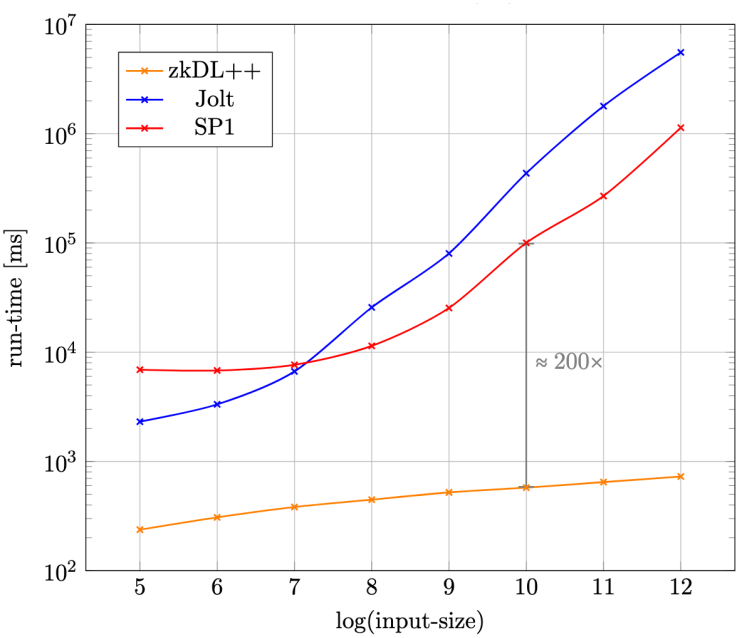

Figure 4: Prover runtime in ms vs. log of input size

the entire network as a whole — zkDL++ strikes a balance between them. This flexibility allows for high computational parallelism while maintaining a manageable commitment cost.

Our proof-of-concept is built on a fork of zkDL. Initially, zkDL only supported primitives for fully connected and ReLU layers. By adding support for convolutional and average pooling layers, we were able to construct a SNARK for the watermark extractor DNN. In the following sections, we present the run-time results of our proof-of-concept, delve into the mathematical foundations of the proving primitives, and conclude with a discussion on limitations and future directions.

3.1 Results and Methodology

We run an end-to-end SNARK prover to prove the correctness of the inference of the extractor DNN. The computation is split into the following steps.

1. Preprocessing: We commit to the parameters of the DNN, i.e. the weights and biases of the trained DNN model. Once the model is trained, this is a one-time computation.

2. Inference: We run the inference on randomized input (of size $3 \times 128 \times 128$) with the extractor DNN. In the context of SNARKs, this step is referred to as witness generation.

3. Witness commitments: We start generating the proof by first committing to the witnesses. Specifically, we commit to the input and output of each layer individually.

4. Sumcheck proofs: We then compute the sumcheck proofs for each layer that attests to the correctness of each layer individually. This is followed by the computation of opening proofs for the witness polynomials in each layer.

We benchmark the zkDL++ framework on the same hardware as described in the previous section. Table 1 shows the run-times of each of these steps.

The total time for generating the proof is approximately 5.4 minutes. Next, we discuss the overview of our techniques. The objective is to arithmetize the primitives used in DNNs.

Table 1: Proof run time

COMPUTATION	TIME [sec]
Preprocessing	5.1
Inference	0.5
Witness commitments	143.5
Sumcheck proofs	172
Total time	321.1

3.2 CONVOLUTION

To keep things concise, we'll focus on the arithmetization of 1D convolution. Let u and v be vectors of length n representing the input and output of the convolution, respectively, and let f be a filter of length $2t + 1$.

$$v(i) = \sum_{j=0}^{2t} f(j)u(i + t - j), \quad i = 0, \ldots, n - 1 \tag{1}$$

We aim to represent this convolution between the input u and the filter f as a polynomial equation. Let $p_u(x)$ and $p_f(x)$ be the polynomials corresponding to the coefficients of u and f, respectively as

$$p_u(x) = \sum_{k=0}^{n-1} u(k)x^k, \qquad p_f(x) = \sum_{k=0}^{2t} f(k)x^k \tag{2}$$

Multiplying these yields the result of the convolution, together with some tail terms

$$p_u(x)p_f(x) = \sum_{k=0}^{n+2t-1} \left(\sum_{j=0}^{2t} f(j)u(k - j) \right) x^k \tag{3}$$

$$= \Delta_1(x) + x^t p_v(x) + x^{n+t}\Delta_2(x) \tag{4}$$

where $p_v(x)$ is the polynomial representing the output vector v, and $\Delta_1(x), \Delta_2(x)$ are the tail terms. The prover needs to demonstrate that this relationship between $p_u(x), p_f(x)$ and $p_v(x)$ holds at a random point $x = r$ chosen by the verifier.

We use the sumcheck protocol to prove this relationship. However, the main challenge for the prover lies in sending evaluations of univariate polynomials, while the sumcheck protocol typically handles multivariate polynomials. To address this, we note that evaluating $p_u(r)$ for $r \in \mathbb{F}$ can be interpreted as an inner product of the vector u with the vector $\vec{r} = (1, r, r^2, \ldots, r^{n-1})$ as

$$p_u(r) = \sum_{k=0}^{n-1} u(k)r^k = \langle u, \vec{r} \rangle \tag{5}$$

The sumcheck protocol can now be applied to verify this inner product. The prover uses this approach to compute the evaluations $p_u(r), p_f(r)$ and $p_v(r)$, while the verifier checks that the relationship holds. Note that the prover can send the tail polynomials Δ_1, Δ_2 directly to the verifier, as their size is relatively small.

This analysis can be trivially extended to the case of 2d convolution. For 2d convolution, we multiply the bi-variate polynomials $p_u(x, y), p_f(x, y)$ to get the convolution output polynomial $p_v(x, y)$. In

this case, if we view the coefficients of $p_u(x,y) \cdot_f (x,y)$ as a matrix where the (i,j) entry is the coefficient for $y^i x^j$ then the output polynomial coefficients are present in the center of the matrix, as shown in figure 5. Notice that we get 8 tail terms.

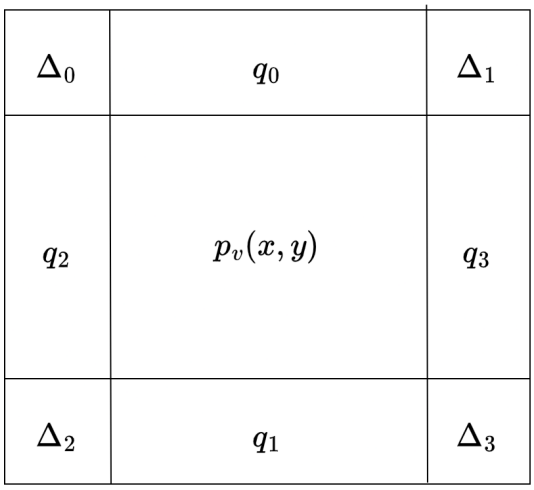

Figure 5: Coefficients of $p_f(x,y)p_u(x,y)$ represented as a 2d matrix: (i,j) entry is the coefficient for $y^i x^j$.

3.3 Average Pooling

To arithmetize the average pooling layer, we need to represent it as a sumcheck instance. Let u be the input of size d, and v the output of size D of the average pooling layer. The output is defined for each $i \in \{1, \ldots, d\}$ as

$$v_i = \frac{1}{s} \cdot \sum_{j=1}^{s} u_{si+j} \tag{6}$$

where $s = D/d$ is the subsampling factor (i.e. the size of the filter). This operation can be represented as the following matrix-vector product

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix} = \begin{bmatrix} \frac{1}{s} & \cdots & \frac{1}{s} & & & & & & \\ & & & \frac{1}{s} & \cdots & \frac{1}{s} & & & \\ & & & & & & \ddots & & \\ & & & & & & & \frac{1}{s} & \cdots & \frac{1}{s} \end{bmatrix}_{d \times D} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_D \end{bmatrix} \tag{7}$$

Matrix multiplication can be efficiently proven using sumcheck as demonstrated by Thaler (2013). Therefore, the correctness of the average pooling operation can be verified using sumcheck. However, a minor challenge arises as the verifier would need to evaluate a bi-variate polynomial, corresponding to the constant matrix in the above equation, on a challenge $r_y \in \mathbb{F}^D$, which implies linear work for the verifier. To avoid this and ensure efficient verification, we can exploit the structure of the matrix in (8) as $I_{d \times d} \otimes \begin{bmatrix} \frac{1}{s} & \ldots & \frac{1}{s} \end{bmatrix}_{1 \times s}$. The prover can leverage this tensor product structure to construct a more efficient sumcheck since it is essentially a multiplication of two multi-linear polynomials in different variables.

3.4 FULLY CONNECTED

A fully-connected layer applies a weight matrix on an input vector to generate an output vector. Let u be the input vector and v be the output vector, then this operation is represented as a matrix-vector product as

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \tag{8}$$

As noted in the average pooling case, we can use the sumcheck technique for matrix multiplication to prove the correctness of the fully-connected layer. Note that the weight matrix is committed to as a part of pre-processing.

3.5 RELU

The zkDL paper gives a technique to arithmetize the ReLU function as a sumcheck instance. Since ReLU is a non-linear operation, we need a slightly different approach in its arithmetization. For the purpose of exposition, we will demonstrate arithmetization for a single ReLU function. Let u be the input to the ReLU function and v be its output. The ReLU function is applied on a scaled version of the input as follows

$$v = \mathsf{ReLU}\left(\left\lfloor \frac{u}{2^R} \right\rceil\right).$$

where R is a constant and $\lfloor \cdot \rceil$ is the rounding operator. Let Q_u be the quotient and R_u be the remainder when we divide u by 2^R . Thus, we have

$$u = 2^R \cdot Q_u + R_u. \tag{9}$$

By definition R_u must be an R-bit integer. Let Q_u be a Q-bit signed integer defined as

$$Q_u = (B_{Q-1} \parallel B_{Q-2} \parallel \dots \parallel B_0) \tag{10}$$

where the most significant bit B_{Q-1} encodes the sign of Q_u (0 if Q_u is positive and 1 if Q_u is negative). Then, by definition of the ReLU function, we have

$$v = (1 - B_{Q-1}) \cdot Q_u. \tag{11}$$

Therefore, to prove the correctness of the ReLU layer, we must prove that equations (9, 10 and 11) hold, along with the condition that R_u is an R-bit integer. These conditions should suffice to prove that the ReLU layer computation was correctly performed.

To prove the bit decomposition of the integer Q_u, we can use sumcheck to prove the following relations

$$Q_u = -B_{Q-1}2^{Q-1} + \sum_{i=0}^{Q-2} B_i 2^i, \quad \text{and} \quad \forall i : B_i(B_i - 1) = 0 \tag{12}$$

Similarly, we must prove that the bit decomposition of the remainder R_u is correct and that $R_u < 2^R$. For all neurons across a layer $j = 1, \dots, n$, We prove the relations between the quotient $Q_u^{(j)}$ and the output $v^{(j)}$ by batching them together into a single sumcheck

$$\sum_{j=1}^{n} \mathsf{eq}(\tau, j)(v^{(j)} - (1 - B_{Q-1}^{(j)})Q_u^{(j)}) = 0 \tag{13}$$

All of the different sumcheck relations are then batched into a single sumcheck for the entire ReLU layer.

4 LIMITATIONS AND FUTURE WORK

The current proof generation process, taking approximately 5.4 minutes, can appear resource-intensive when scaled for content production. A significant portion of this time is spent on two key tasks: computing witness commitments and generating sumcheck proofs. At present, these operations are performed sequentially, which introduces inefficiencies that can be addressed.

1. Sequential Witness Commitments: Proof generation involves computing commitments for 13 witness vectors, corresponding to 9 convolution layers, 1 average pooling layer, 1 fully connected layer, and the input and output. These commitments are currently calculated layer by layer in a linear fashion, starting from the input and progressing to the output. This sequential process creates bottlenecks, particularly when dealing with large models. Parallelizing the computation of these witness commitments is a straightforward and promising area for improvement.
2. Batching Sumcheck Proofs: Each layer also requires the generation of sumcheck proofs, which independently verify different properties of that layer. Presently, sumcheck proofs are computed one after the other, further contributing to the extended proof generation time. There is scope to parallelize the sumcheck proofs across layers, as they are independent. Additionally, within each layer, multiple sumcheck proofs are needed to prove various statements. Batching these sumcheck computations — both within individual layers and across different layers — can help reduce redundancy and improve overall performance.
3. Choice of commitment scheme: The current implementation uses the Hyrax multi-linear commitment scheme for witness commitments and sumcheck opening proofs. While Hyrax is effective, exploring alternative schemes such as Zeromorph (Kohrita & Towa, 2023) could be beneficial, particularly for applications where evaluation proofs need to be zero-knowledge. Zeromorph offers potential advantages in terms of performance and security in zero-knowledge settings.
4. Choice of the field: The current setup uses the BLS12–381 elliptic curve and its corresponding scalar field for computations. While this curve is widely adopted for cryptographic purposes, exploring alternative fields, such as Mersenne primes or Baby Bear (and their extensions), could provide speedups in the proof generation process.

There are several promising avenues for improving the efficiency of proof generation. Parallelizing witness commitments and sumcheck proofs, exploring alternative commitment schemes, and experimenting with smaller fields represent significant opportunities for future optimization.

4.1 WHAT'S NEXT?

In this work, we demonstrated zkDL++ with a real-world example involving a CNN. However, the zkDL++ framework is flexible and can be extended to support additional layers, making it applicable to any deep neural network (DNN). In our proof of concept for private watermark extraction, we achieved prover runtimes in the range of minutes for a standard CNN. With further optimizations, we believe this can be reduced to seconds, meeting the performance requirements for large-scale deployment by companies like Meta.

Our code is production-ready and available for testing. We encourage collaborators, researchers, and developers to explore zkDL++ with us and help advance this powerful framework.

ACKNOWLEDGMENTS

We would like to thank the authors of the zkDL and Stable Signature papers. Special thanks goes to Pierre Fernandez and Haochen Sun.

REFERENCES

Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: SNARKs for virtual machines via lookups. Cryptology ePrint Archive, Paper 2023/1217, 2023. URL https://eprint.iacr.org/2023/1217.

Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. VIMz: Private proofs of image manipulation using folding-based zkSNARKs. Cryptology ePrint Archive, Paper 2024/1063, 2024. URL https://eprint.iacr.org/2024/1063.

Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. The stable signature: Rooting watermarks in latent diffusion models, 2023. URL https://arxiv.org/abs/2303.15435.

Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), September 2015. ISSN 0004-5411. doi: 10.1145/2699436. URL https://doi.org/10.1145/2699436.

Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023. URL https://eprint.iacr.org/2023/917.

Succinct Labs. Sp1, 2024. URL https://github.com/succinctlabs/sp1.

Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. Cryptology ePrint Archive, Paper 2021/673, 2021. URL https://eprint.iacr.org/2021/673.

Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992. ISSN 0004-5411. doi: 10.1145/146585.146605. URL https://doi.org/10.1145/146585.146605.

Robin San Roman, Pierre Fernandez, Alexandre Défossez, Teddy Furon, Tuan Tran, and Hady Elsahar. Proactive detection of voice cloning with localized watermarking, 2024. URL https://arxiv.org/abs/2401.17264.

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. URL https://eprint.iacr.org/2023/1174.

Justin Thaler. Time-optimal interactive proofs for circuit evaluation. Cryptology ePrint Archive, Paper 2013/351, 2013. URL https://eprint.iacr.org/2013/351.