Provable Watermark Extraction

Suyash Bagad [1] Yuval Domb [1] Ashutosh Marwah [2] Omer Shlomovits [1]
Tomer Solberg [1]

[1]Ingonyama [2]University of Montreal

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.

Introduction

Meta AI's Stable Signature is a watermarking (WM) scheme for identifying images generated by latent diffusion models (LDMs). However, it must keep the extractor model private to prevent attacks, making it hard to prove the existence of the WM in case of ownership dispute.

We propose solving this problem using Zero-Knowledge Proofs (ZKP) of the relation $F(X, W) = 0$ where $X = (x_{in}, x_{out})$ is public, and $W = (w, k)$ is a private witness.

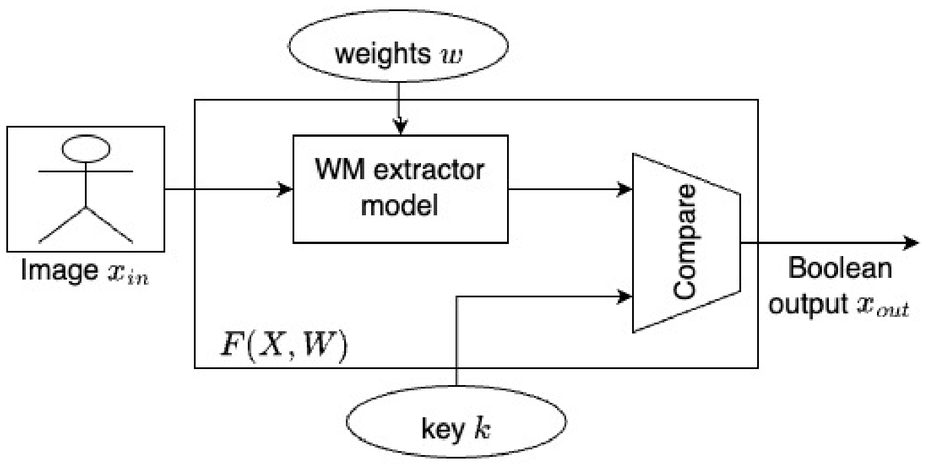

Figure 1. The WM extraction relation

To prevent the prover from choosing weights arbitrarily to forge a valid proof, we incorporate a cryptographic commitment to the weights $C = c(W)$ which is published together with the WM system and added to X in F.

CNN architecture

In Meta's Stable Signature, the watermark extractor is a CNN which takes an image as input, passing it through 9 convolution layers, an average pooling layer, and a fully connected layer, as shown in figure 2, where we assume the image is 128x128 pixels

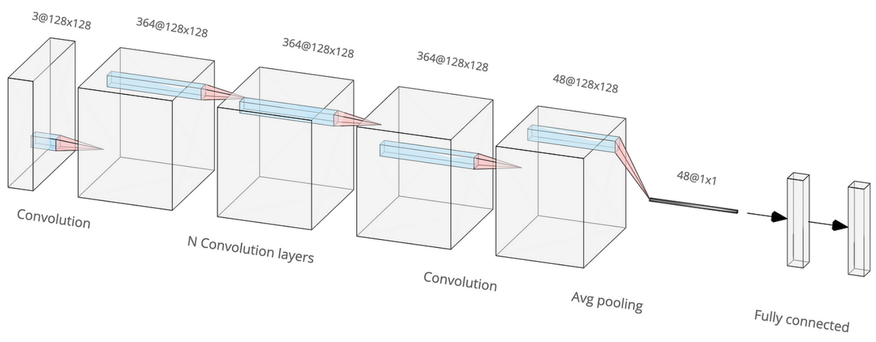

Figure 2. Stable Signature's DNN architecture

zkDL++ 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 (weights and biases) of the trained DNN, this is a one-time computation.
2. Inference (witness generation): We run the inference on input (of size $3 \times 128 \times 128$) with the extractor DNN.
3. Witness commitments: We commit to the input and output of each layer of the witness individually. We use Hyrax PCS.
4. Sumcheck proofs: We compute sumcheck proofs for each layer that attest to the correctness of each layer individually. This is followed by computation of opening proofs for the witness polynomials in each layer.

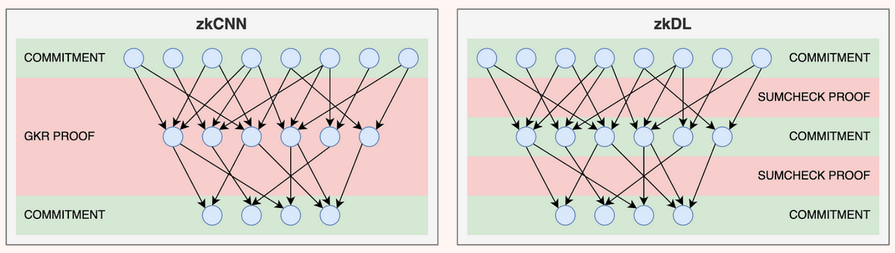

Figure 3. zkCNN vs. zkDL

Convolution

Given an input u of length n and a filter f of length $2t + 1$, we represent the output v of length n using polynomial multiplications, where $p_\alpha(x) = \sum_k \alpha(k)x^k$ for $\alpha = u, f, v$

$$p_u(x)p_f(x) = \sum_{k=0}^{n+2t-1} \left(\sum_{j=0}^{2t} f(j)u(k-j) \right) x^k = \Delta_1(x) + x^t p_v(x) + x^{n+t}\Delta_2(x) \tag{1}$$

Which can be straight-forwardly generalized to 2D. The prover commits to all of these polynomials, and then demonstrates 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, while also running a sumcheck protocol to prove $p_\alpha(r) = \langle u, \vec{r} \rangle$ for all α, where $\vec{r} = (1, r, r^2, \ldots, r^{n-1})$.

Fully connected and average pooling

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

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

As shown by Thaler, sumcheck can be efficiently used to prove matrix multiplication and thus the correctness of the fully-connected layer. Note that the weight matrix is committed to as a part of pre-processing. The same method applies for average pooling, with the length of the output reduced by a factor of s to $d = n/s$, and the matrix is $I_{d\times d} \otimes \left[\frac{1}{s} \ \ldots \ \frac{1}{s}\right]_{1\times s}$.

ReLU

The ReLU function is applied on a scaled version of the input as $v = \text{ReLU}\left(\lfloor u/2^R \rceil\right)$. To arithmetize this into a sumcheck instance, We define

$$u = 2^R \cdot Q_u + R_u, \quad Q_u = -B_{Q-1}2^{Q-1} + \sum_{i=0}^{Q-2} B_i 2^i \tag{3}$$

Where $0 \leq R_u < 2^R$ and $B_i \in \{0, 1\}$. These are all simple relations to prove. The ReLU relation itself is then $v = (1 - B_{Q-1})Q_u$. We can then batch many such relations to prove an entire ReLU layer $v^{(j)},\ j = 1, \ldots, n$ using sumcheck

$$\sum_{j=1}^{n} \text{eq}(\tau, j)(v^{(j)} - (1 - B_{Q-1}^{(j)})Q_u^{(j)}) = 0 \tag{4}$$

Results

We benchmark zkDL++ along with widely used zkVMs 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). 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. Figure 4 compares performance for proving fully connected layers of different sizes

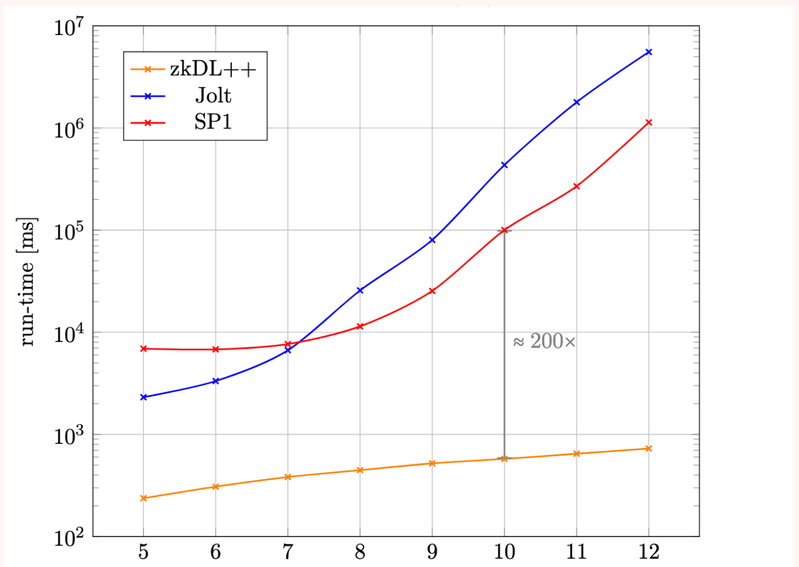

Figure 4. Prover runtime in ms vs. log of input size

The entire proof generation for Meta's DNN is currently done in 5.4 minutes. This is expected to be reduced significantly by future optimizations, such as improved GPU utilization, sumcheck proof batching, better commitment schemes and using smaller fields.

Acknowledgments

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