Bionetta: Ultimate ZKML Framework

April 25, 2025

Distributed Lab
Rarimo

distributedlab.com/
github.com/rarimo/bionetta-tf

Intro to ZKML

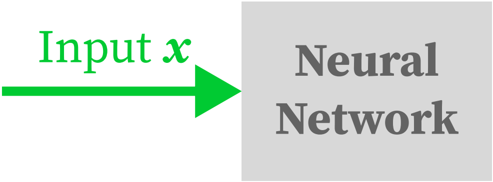
What is a neural network?

Think of a neural network as a **black box**...

Neural Network

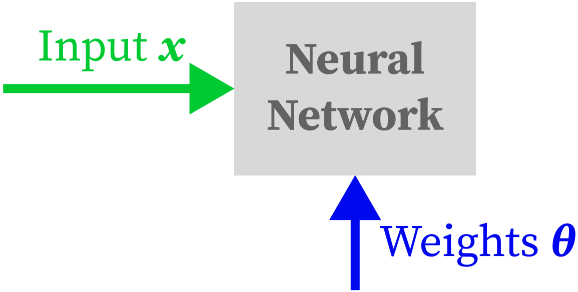
What is a neural network?

This **black box** takes some input x (e.g., image or a text prompt)...



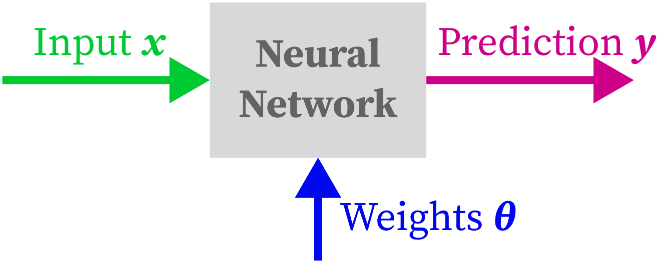
What is a neural network?

Besides the input x, you can tweak the parameters θ of the black box — so-called *weights* — which changes the behavior of the black box. They are typically *fixed*...



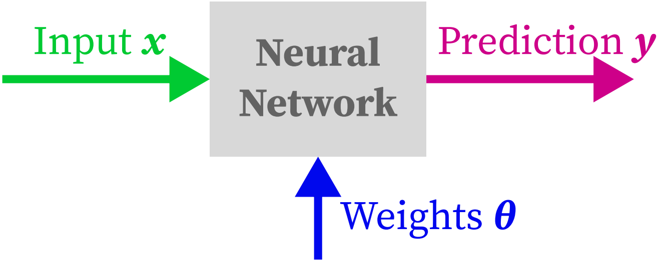
What is a neural network?

Given the inputs x and weights θ, you can get the prediction y (e.g., person's features or AI's text response)...



What is a neural network?

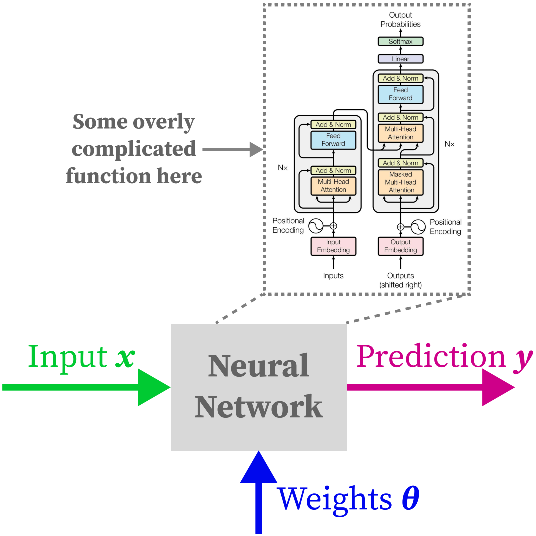
Given the inputs $\boldsymbol{x}$ and weights $\boldsymbol{\theta}$, you can get the prediction $\boldsymbol{y}$ (e.g., person's features or AI's text response)...



We denote such computation as $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$.

What is a neural network?

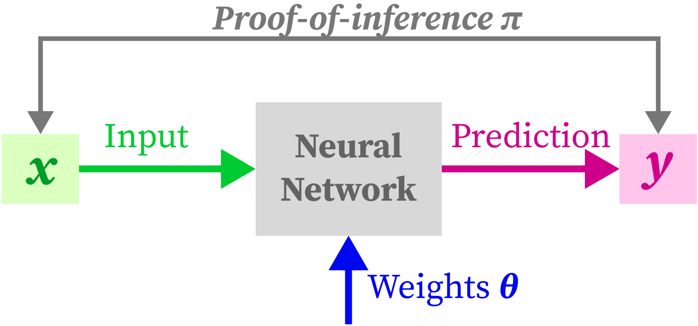
Though, in practice everything is *quite complicated*...



Where ZK?

So why do we need ZK in this process?

ZKML: Engineering Perspective



Prove that for $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}$ we indeed have $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$.

Yet, what is public and what is private? Obviously, $\boldsymbol{y}$ is public, so what about $\boldsymbol{x}$ and $\boldsymbol{\theta}$?

ZKML: Private Weights θ, Public Input x

User

Give me my bank card →

OpenAI

ZKML: Private Weights θ, Public Input x

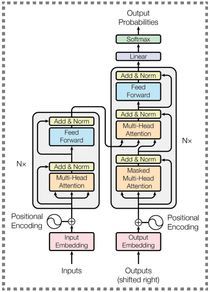
User

Give me my bank card

OpenAI

Predict

Neural Network



ZKML: Private Weights θ, Public Input x

User

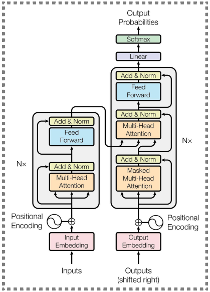
Give me my bank card

Sure! Here it is **Card A**

OpenAI

Predict

Neural Network



ZKML: Private **Weights** θ, Public **Input** x

User

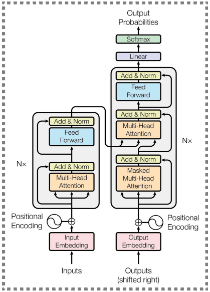
Give me my bank card

Sure! Here it is **Card B**

OpenAI

Predict

Neural Network



ZKML: Private **Weights** θ, Public **Input** x

User

x = **"Give me my bank card"**

OpenAI

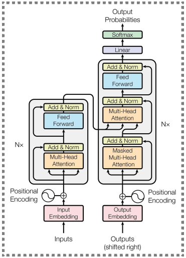
ZKML: Private Weights θ, Public Input x

User

x = “Give me my bank card”

OpenAI

Predict



Neural Network

ZKML: Private Weights θ, Public Input x

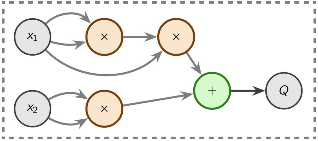
User

x = **"Give me my bank card"**

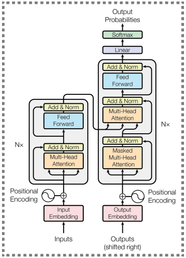
OpenAI

Predict

Verify **Prove**



Neural Network Circuit



Neural Network

ZKML: Private Weights θ, Public Input x

User

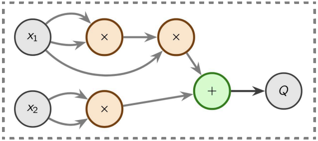
x = "Give me my bank card"

OpenAI

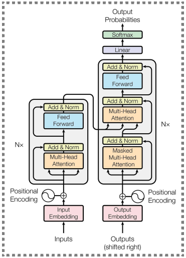
Proof π

Predict

Verify **Prove**

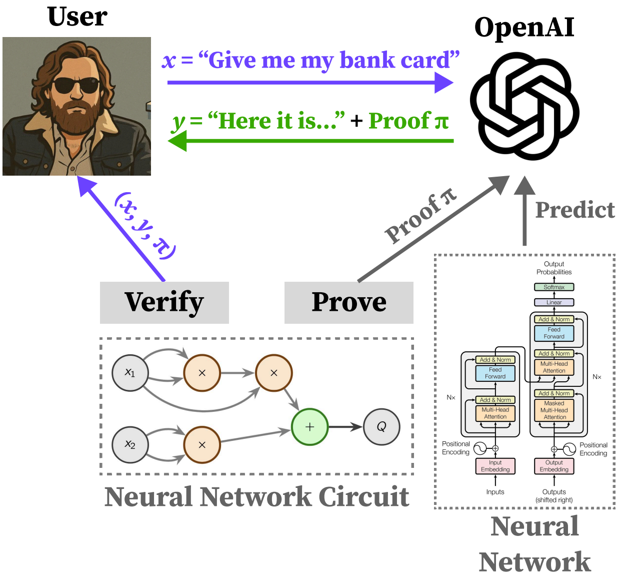


Neural Network Circuit



Neural Network

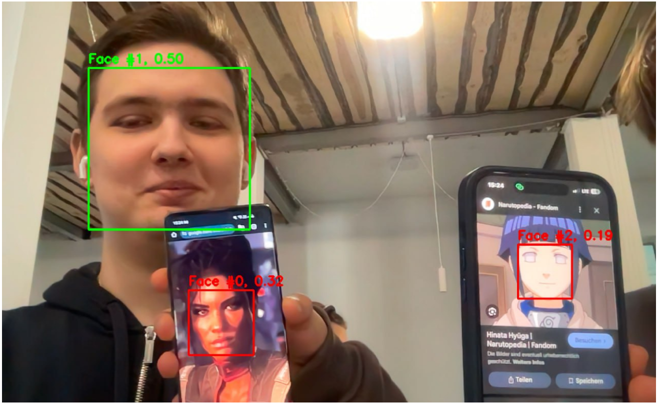
ZKML: Private Weights θ, Public Input x



Where ZK? Public Weights θ, Private Input x

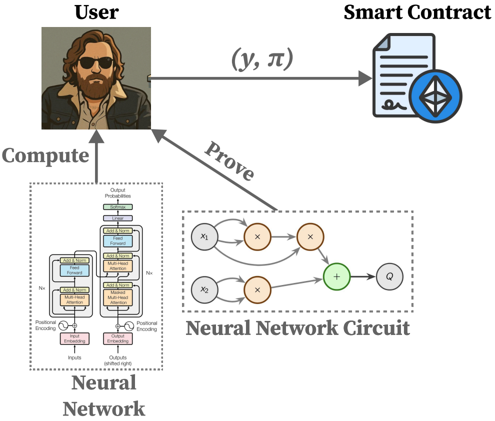
Problem: private weights $\implies$ private model $\implies$ centralization.

So the **client-side** is the only way to go!



Our Usecases

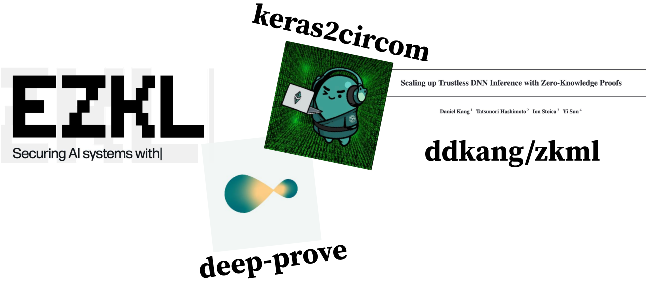
- ✓ **Biometric Proximity Proof:** proof that you are you based on the biometric data (e.g., face, fingerprint, etc.).
- ✓ **Liveness Proof:** proof that you are indeed an alive human being (e.g., not a bot) based on the screenshot.



Benchmarks

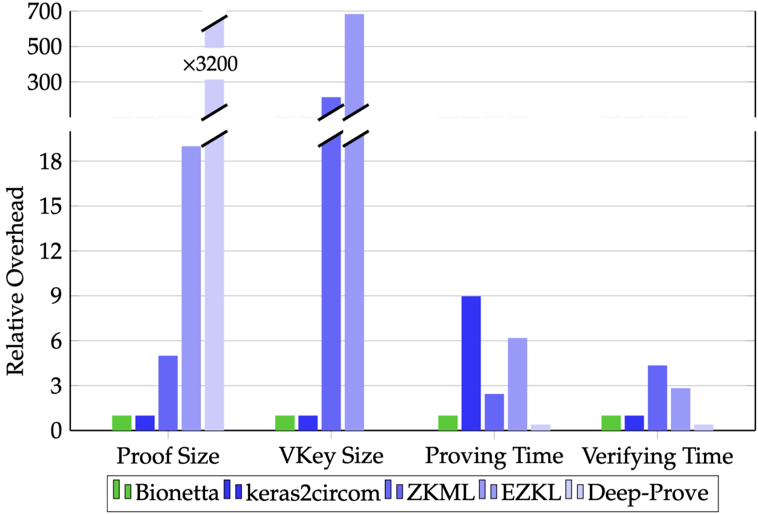
Client-Side ZKML Requirements

- ✓ **Fast:** the proof generation time should be less than a minute.
- ✓ **Lightweight:** the proof must be small enough to be provable on the smart-contracts (without significant fee increase). Verification key should also be small for similar reasons.
- ✓ **Not resource-consumable:** the proof generation should be manageable on small devices.



Bionetta Benchmarks

https://rarimo.com/learning-hub/benchmarking-bionetta-61



Bionetta Benchmarks in the Wild

- ✓ **Liveness Neural Network** is roughly **1.5 mln parameters** in size, takes **1 million constraints** and roughly **20 seconds** and **1.5GB** of RAM to generate a proof. Benchmark accuracy is 96%.
- ✓ **Face Recognition Neural Network** is roughly **2.0 mln parameters** in size, takes **800k constraints**. Currently measuring the time, but expected to be similar to the Liveness NN.

Note

We haven't tested running *Bionetta* over existing neural network (e.g., *MobileNetV2*) and currently use customly-crafted NNs.

If the neural network contains N non-linearity calls, then the circuit size $|C|$ can be approximated as $\boxed{|C| \approx 255N}$.

Developing Bionetta

Step I: Train the Model

Preferrably, on the Bionetta custom layers:

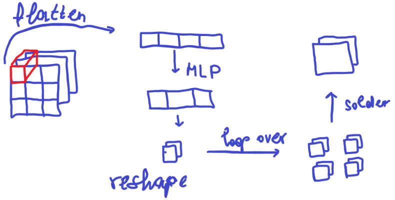


Figure: EDLightConv2D

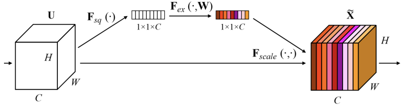
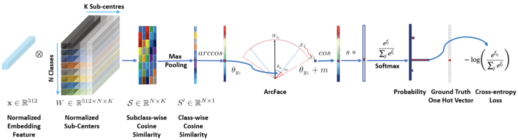


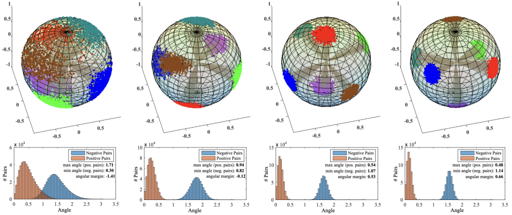
Figure: SELightBlock

Training is Hard!



$$L_4 = -\log \frac{e^{s(\cos(m_1\theta_{y_i}+m_2)-m_3)}}{e^{s(\cos(m_1\theta_{y_i}+m_2)-m_3)} + \sum_{j=1,j\neq y_i}^{N} e^{s\cos\theta_j}}. \quad (4)$$

As shown in Figure 4(b), by combining all of the above-motioned margins ($\cos(m_1\theta + m_2) - m_3$), we can easily get some other target logit curves which also achieve high performance.



A-Softmax (m=1) A-Softmax (m=2) A-Softmax (m=3) A-Softmax (m=4)

Thus we want,

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 , \quad (1)$$

$$\forall\,(f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T} . \quad (2)$$

where α is a margin that is enforced between positive and negative pairs. $\mathcal{T}$ is the set of all possible triplets in the training set and has cardinality N.

The loss that is being minimized is then $L =$

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+ . \quad (3)$$

Underfit, Vanishing Gradients, Vector Collapse etc. . .

Step II: Compiling Circuits

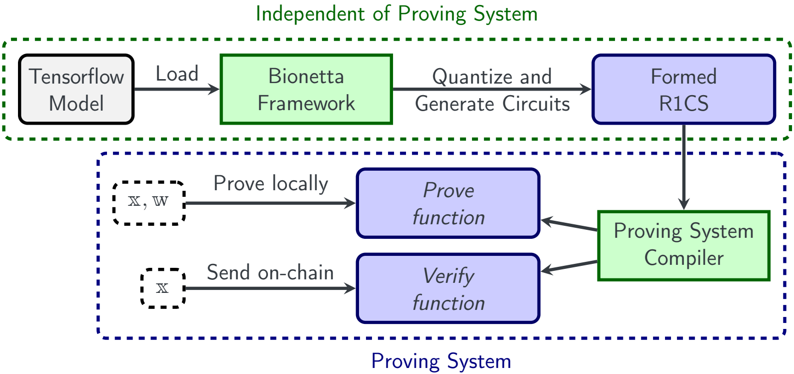


Figure: Architecture of the Bionetta framework

Note

Our *BionettaV1* framework is, in fact, an **R1CS constructor**.

Architecture: Low-Level

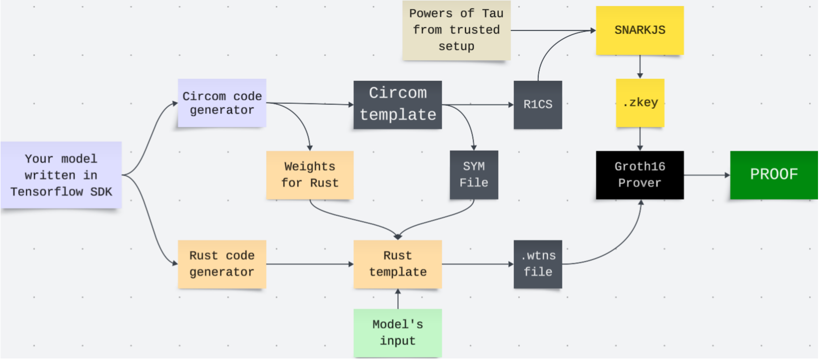


Figure: Low-Level Bionetta Architecture

TF Model → Bionetta → Circom → R1CS → Rust → Bindings

We've built our custom (blazingly) fast Rust **witness generator**!

Key Optimization: Circuit-Embedded Weights

Assume you want to implement a function:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{i=1}^{n} \theta_i x_i + \theta_0 \quad \texttt{// Linear Regression}$$

Idea #1
Public Signal: Weights $\boldsymbol{\theta} = (\theta_0, \dots, \theta_n)$.
Private Signal: Inputs $\boldsymbol{x} = (x_1, \dots, x_n)$.
Circuit: First, assert $r_i = \theta_i x_i$ for each $i \in \{1, \dots, n\}$. Then compute the result $\theta_0 + \sum_{i=1}^{n} r_i$. **Circuit size:** $\mathcal{O}(n)$.

Idea #2
Constants: Weights $\boldsymbol{\theta} = (\theta_0, \dots, \theta_n)$.
Private Signal: Inputs $\boldsymbol{x} = (x_1, \dots, x_n)$.
Circuit: Compute linear sum $\sum_{i=1}^{n} \theta_i x_i$ directly. **Circuit size:** 0.

Corollary: Circuit-Embedded Weights

Corollaries

- ✓ The majority of the traditional Machine Learning algorithms such as PCA, LDA, linear or logistic regression costs **0 constraints**.
- ✓ All linear operations inside the neural network are free.

Corollary: Circuit-Embedded Weights

Corollaries

✓ The majority of the traditional Machine Learning algorithms such as PCA, LDA, linear or logistic regression costs **0 constraints**.

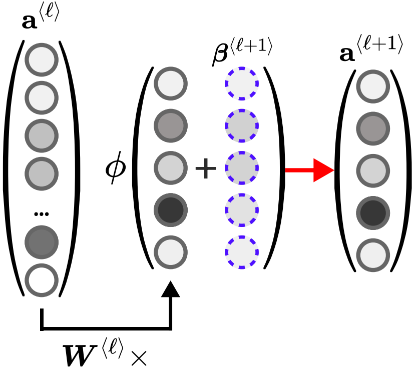
✓ All linear operations inside the neural network are free.

Conclusion. We must use **R1CS-compatible** proving framework such as: Groth16, Spartan, Ligero, Aurora, Fractal.

Corollary: Circuit-Embedded Weights

Corollaries

- ✓ The majority of the traditional Machine Learning algorithms such as PCA, LDA, linear or logistic regression costs **0 constraints**.
- ✓ All linear operations inside the neural network are free.



Issue. Typically, after the linear operations, we apply the non-linear operation (e.g., $\max\{0, x\}$). Each one currently costs **255 constraints**. We have an approach to reduce this cost down to ≈ 20 constraints.

Problems

✓ **Problem 1.** Add support for more neural network layers.

✓ **Problem 2.** Activation-optimized neural networks are *very hard* to train. Further optimizations allow more complex models $\implies$ better accuracy. E.g., 1 mln constraints $\approx$ 3900 non-linearities.

Problems

✓ **Problem 1.** Add support for more neural network layers.

✓ **Problem 2.** Activation-optimized neural networks are *very hard* to train. Further optimizations allow more complex models $\implies$ better accuracy. E.g., 1 mln constraints $\approx$ 3900 non-linearities.

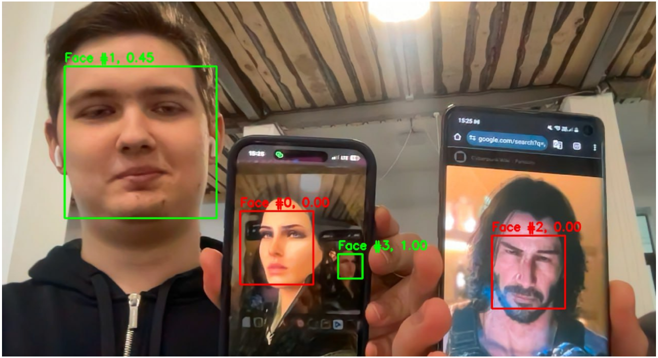


Figure: Shit happens

Problems

- ✓ **Problem 1.** Add support for more neural network layers.
- ✓ **Problem 2.** Activation-optimized neural networks are *very hard* to train. Further optimizations allow more complex models $\implies$ better accuracy. E.g., 1 mln constraints $\approx$ 3900 non-linearities.

Future Directions

Goal 1. Implement **UltraGroth**: $\approx$ 20 gates per activation.

- Compute the final commitment $C^{(d)}$:

$$C^{(d)} = \sum_{j \in \text{round}_d} w_j C_j + \sum_k h_k Z_k + sA + rB' - \sum_{k<d} r_k [\delta_k]_1 - rs[\delta_d]_1$$

- Compute the public input commitment:

$$\text{IC} = \sum_{j \in \text{pub}} w_j C_j$$

Verification

The verifier performs the pairing check:

$$e(A, B) = e([\alpha]_1, [\beta]_2) \cdot e(\text{IC}, [\gamma]_2) \cdot \prod_{k=0}^{d} e(C^{(k)}, [\delta_k]_2)$$

Figure: Excerpt from our UltraGroth technical specification.

Can be used in other projects as well if you need effective range checks!

Future Directions

Goal 2. More blogs and media activity on the way.

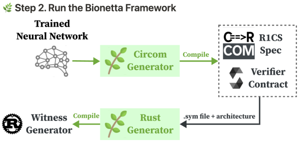
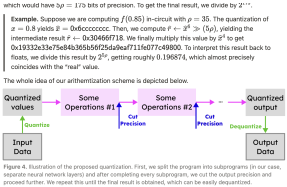


Figure: Engineering Blogs



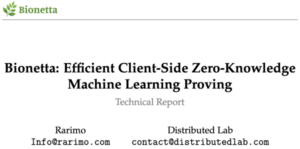
which would have $5\rho = 175$ bits of precision. To get the final result, we divide by $2^{[illegible]}$.

Example. Suppose we are computing $f(0.85)$ in-circuit with $\rho = 35$. The quantization of $x = 0.8$ yields $\widehat{x} = $ 0x6ccccccc. Then, we compute $\hat{r} \leftarrow \widehat{x}^6 \gg (5\rho)$, yielding the intermediate result $\hat{r} \leftarrow$ 0x30466f718. We finally multiply this value by $\widehat{x}^4$ to get 0x19332e33e75e84b365b56f25da9eaf711fe077c49800. To interpret this result back to floats, we divide this result by $2^{5\rho}$, getting roughly 0.196874, which almost precisely coincides with the "real" value.

The whole idea of our arithemtization scheme is depicted below.

Figure 4. Illustration of the proposed quantization. First, we split the program into subprograms (in our case, separate neural network layers) and after completing every subprogram, we cut the output precision and proceed further. We repeat this until the final result is obtained, which can be easily dequantized.

Figure: Technical Blogs



Bionetta

Bionetta: Efficient Client-Side Zero-Knowledge Machine Learning Proving

Technical Report

Rarimo
Info@rarimo.com

Distributed Lab
contact@distributedlab.com

Figure: Research Papers

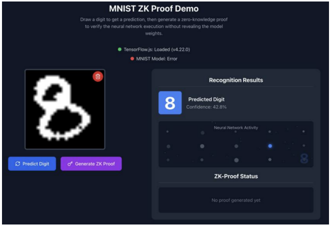


Figure: Demos

Future Directions

Goal 3. Noir + Bionetta Integration.

In particular, this includes:

- ✓ Custom ACIR to R1CS converter.
- ✓ Groth16 (*and potentially UltraGroth*) backend.
- ✓ Circuits written in a human-readable format (yes, Circom, we are looking at you).

Any Questions?

♥

distributedlab.com
github.com/distributed-lab/nero