By Todd Stool

EVMAI
ERC-20AI
EVMAI
ERC-20AI
EVMAI

The AI on Blockchain (EVM) Perspectives

The Introduction.

AI enhances our daily lives across various experiences. The concept of AI-driven tokens represents a logical step forward in the evolution of blockchain and banking systems. To make this vision a reality, AI must either be deployed fully on-chain or operate in a hybrid model with partial off-chain processing. However, off-chain elements introduce vulnerabilities, making AI hosted directly on the Ethereum Virtual Machine (EVM) a more secure and viable solution for the near future.

The concept of ERC-20ai serves as a proof of concept for such technology. Imagine an AI that is fully deployed on-chain, benefiting from the EVM's built-in resistance mechanisms and verifiability. While such a deployment is currently impractical on Ethereum due to limitations in gas costs and processing power, it is a possibility for the future as infrastructure evolves.

On-chain AI could serve a wide range of purposes, from managing smart contracts autonomously to generating images such as AI-generated NFTs, and even producing unique content directly on-chain. This content could have clearly defined ownership to empower and protect creators.

Introducing the first AI deployed on an EVM-compatible chain: a neural network that generates images from input code, entirely on-chain!

Neutants

The Neutant.

The first-born children of ERC-20AI to pave the way for many more previously impossibly unique concepts. They are blessed with the lore of ruling their AI-generated domain in creepy yet somehow still endearing ways.

The token.

This project consists of two parts: the token itself and the AI logic that powers it. The token has a limited supply of 10,000 units, each with nine decimal places. Every whole token encodes a unique set of six numbers, such as: 3 2 1 4 5 6. These numbers serve as input data for an on-chain AI model which generates a corresponding image. The result is a unique AI-generated artwork that is permanently tied to each token and cannot be replicated.

The "sauce".

To preserve the uniqueness and value of the AI-generated avatars, the system includes a formula that decreases the chance of generating visually coherent images after the first 10,000 have been created which represent the genesis set. From there, a distortion factor, called seed_addition, is applied to subsequent generations. As this parameter increments over time, the more fragmented or surreal the output becomes and the results can range from useless to breathtaking. This 'sauce" ensures that early avatars remain collectible, while later ones have the potential to grow increasingly rare with vivd structure and clarity

The AI.

I used a simple Convolutional Neural Network (CNN) to generate the images on-chain. The CNN was ported to Solidity and deployed on the blockchain. I will explain the AI part on the next page.

Some of the early work in progress images:

The Idea

What is a CNN?

A Convolutional Neural Network (CNN) is a type of neural network designed to process data with a grid-like structure, such as images. Unlike traditional fully-connected neural networks, CNNs are optimized for recognizing spatial patterns by applying filters (also called kernels) across input data.

These filters slide across the image, detecting low-level features like edges or textures in early layers, and more complex patterns (such as shapes or structures) in deeper layers. This hierarchical feature detection makes CNNs especially effective for tasks like image classification, object recognition, and in this case, asset generation.

In my project, I trained a CNN on thousands of pixel art avatars, called Neutants. The model learned to represent visual patterns in the form of weights and biases. I then ported the trained CNN into Solidity and deployed it on-chain. When a token's six-number code is passed as input, the CNN processes it through its layers to produce a unique 24x24 pixel image fully on the EVM.
The result is a compact and efficient model, simplified enough to operate within the gas and performance constraints of on-chain execution, yet robust enough to produce varied and expressive outputs.

The Technical Concept.

To bring this project to life, I first ported the AI model to C# to ensure it functioned correctly. Once everything worked as expected, I began translating it into Solidity for on-chain deployment. The entire development process took about six months. The most challenging and time-consuming part was training the Convolutional Neural Network (CNN). After many failed attempts, I even paused the project for a month to dive deeper into AI concepts, at times doubting whether the goal was achievable at all. But persistence paid off, and I finally reached the results I had envisioned.

The initial concept was to generate high-resolution images, but that approach failed due to EVM limitations. As a result, I reduced the final image size to 24×24 pixels, which still represents a major achievement for fully on-chain AI image generation.

During the testing phase, I built a custom engine to simulate thousands of transactions on a local testnet. This helped uncover potential bugs and edge cases under realistic conditions. It was a long and challenging journey, and I hope users enjoy the results as much as I enjoyed building them.

Before writing the complex algorithms for the blockchain and to ensure that the exported parameters from the PyTorch-trained neural network were working correctly, I built a complete neural network from scratch in C#. After thoroughly understanding the internal workings of the Python-based model and confirming that the generation results matched, I was confident everything was functioning as intended. Only then did I proceed with the blockchain porting, which required implementing algorithms that execute through multiple transactions due to EVM constraints.

The NeuralNetwork Contract.

CNN Architecture and On-Chain Execution

The Convolutional Neural Network (CNN) consists of 8 sequential layers, combining both trainable and non-trainable components:

- *Linear layer*
- *ReLU activation*
- *Linear layer*
- *ReLU activation*
- *Deconvolution layer*
- *ReLU activation*
- *Deconvolution layer*
- *Final ReLU activation (used for optimization and output shaping)*

Out of these, four layers are trainable (two linear and two deconvolution layers), while the other four are non-parametric ReLU activations that add non-linearity to the network.

Due to EVM gas limits, it is not feasible to run the entire network within a single transaction. To solve this, I developed a specialized smart contract architecture that supports incremental computation over multiple transactions. This allows the model to maintain intermediate state on-chain and continue processing without exceeding gas constraints.

Model Training and Parameter Initialization

The network was trained using PyTorch, a powerful deep learning framework. After training, all weights and biases were exported using a custom tool and transferred to Solidity contracts via a series of setter functions:

- *initialize_arrays* – allocates memory and prepares internal structures

- *set_fc_bias_0*, *set_fc_bias_2* – biases for fully connected layers

- *set_fc_weight_0*, *initialize_fc_weight_2*, *set_fc_weight_2* – weights for fully connected layers

- *set_bias_0*, *set_bias_2* – biases for deconvolutional layers

- *set_weight_0*, *set_weight_2* – weights for deconvolutional layers

To avoid exceeding block gas limits during deployment, all parameter uploads are done in small parts. Each function call sets only a subset of data, and the contract keeps track of progress to ensure consistency. This makes the initialization process transparent, safe, and fully verifiable.

The NeuralNetwork Contract.

On-Chain Inference Flow

Once all parameters are set, the network is ready for on-chain inference. Image generation is performed in a series of step-by-step function calls:

forward1 - first linear layer with ReLU

forward2 - second linear layer with ReLU

deconv_iterational_1, *deconv1*, *calc_deconv_sublayer_expand_1* - first deconvolution stage

deconv_iterational_2, *deconv2*, *calc_deconv_sublayer_expand_2* - second deconvolution stage

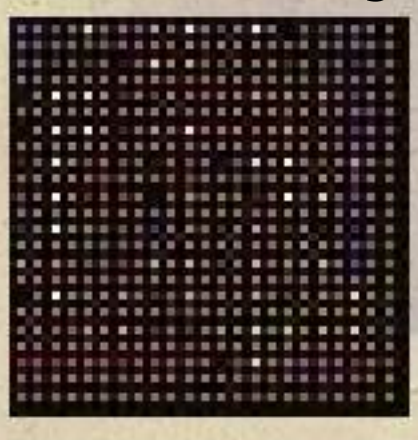
These functions are executed across multiple read transactions, with persistent state between calls. This method allows complex AI inference to happen directly within the blockchain, without requiring any off-chain computation.

JavaScript Library and Website Integration

To make on-chain image generation more accessible, I developed a dedicated JavaScript library that mirrors the contract logic. It interacts with the deployed smart contract, retrieves intermediate data, and reconstructs the final 24×24 image from the neural network's output.

This library powers the project's website, allowing users to visualize their AI-generated avatars in real time based on token input. It serves both as a frontend interface and a developer tool for exploring the power of on-chain AI.

The first images I could achieve during the early development stages:




POST SCRIPTUM

The project is available on Uniswap V3 and www.inscriptions.market for OTC deals.

Always yours, Todd Stool