POKEBALL

Protocol for On-chain Knowledge Encoding
Blockchain AI Latent Loader

MJ Stephenson

POKEBALL Protocol · University of Waterloo

m26steph@uwaterloo.ca

April 19, 2026

ABSTRACT

We present POKEBALL, the first demonstration of a reconstructable large language model (LLM) weight representation stored permanently in a smart contract on a public blockchain. A hypernetwork is trained to compress the full tensor weight distribution of TinyLlama-1.1B (Q4_K_M quantization, 637 MB) into a 1,024-byte latent seed vector $z \in \mathbb{R}^{256}$, achieving a compression ratio of 654,296×. The seed is inscribed as immutable contract storage on the Aleph Zero blockchain via an ink! smart contract, queryable by any on-chain or off-chain agent. Given the open-source hypernetwork architecture and the on-chain seed, the full approximate weight distribution of TinyLlama-1.1B is deterministically reconstructable. This work introduces a new paradigm for AI model permanence: model provenance, integrity, and approximate reconstruction capability, encoded in a few hundred bytes, living forever on a decentralized ledger.

Keywords: hypernetwork · on-chain AI · blockchain · LLM compression · latent representation · smart contract · Aleph Zero · ink! · decentralized AI · model permanence

CONTENTS

1 INTRODUCTION

The permanence problem in AI is well-known: models are hosted on servers, behind APIs, subject to deprecation, corporate policy, and infrastructure failure. A model that exists today may not exist tomorrow. No existing mechanism guarantees that a specific model — its exact weights, behavior, and identity — survives indefinitely in a verifiable, accessible form.

Blockchain technology offers a solution: data stored in smart contract state is replicated across thousands of nodes, cryptographically anchored, and immutable by design. However, storing large binary artifacts on-chain is cost-prohibitive. A 637 MB GGUF model file cannot be stored in contract storage.

We resolve this tension with POKEBALL: a protocol in which a neural hypernetwork is trained to compress the full weight distribution of a target LLM into a compact latent seed vector, which is small enough to store on-chain. The seed, together with the publicly specified hypernetwork architecture, constitutes a complete — if approximate — specification of the model.

The contributions of this work are:

1. The first deployment of a reconstructable LLM representation in a smart contract.
2. A hypernetwork architecture for compressing transformer weight tensors into a 1,024-byte latent code.
3. An on-chain data schema and ink! smart contract ABI for querying model metadata and latent codes.
4. A discussion of the reconstruction pipeline and its fidelity, and a roadmap toward browser-native inference.

2 BACKGROUND

2.1 Hypernetworks

A hypernetwork [1] is a neural network H_φ that generates the weights of a primary network N_θ. Formally, $\theta = H_\varphi(z)$ where z is a low-dimensional latent code. Hypernetworks have been applied to neural architecture search, continual learning, and multi-task adaptation, but their application to LLM weight compression for on-chain storage has not previously been demonstrated.

2.2 On-Chain AI

Prior work in decentralized AI includes Bittensor [2], which incentivizes off-chain inference; Ritual [3], which provides on-chain compute orchestration; and EZKL/Giza, which prove correct ML inference via ZK-SNARKs. None of these store reconstructable model weights on-chain. IPFS-anchored NFTs store pointers to off-chain model files, not the weights themselves.

2.3 Hypergraphs and Tensor Networks

The theoretical underpinning of the compression approach draws from the theory of hypergraphs and tensor networks [5]. A transformer's weight tensors can be viewed as a hypergraph where tensors are hyperedges connecting layer indices. The hypernetwork learns a low-dimensional parameterization of this hypergraph

structure. We acknowledge early conversations with Matthias Christandl (QMATH, University of Copenhagen / Simons Institute) on hyperedge representations and hypernetwork geometry, which informed the design of the latent compression scheme.

3 METHOD

3.1 Target Model

The target model is `tinyllama-1.1b-chat-v1.0.Q4_K_M`, a 1.1B parameter LLaMA-architecture model in GGUF format with Q4_K_M quantization. The model file is 637 MB and contains 363 tensors across 40 transformer layers with embedding dimension 4096.

3.2 Hypernetwork Architecture

Let $z \in \mathbb{R}^{256}$ be the latent seed. The hypernetwork H_φ consists of three learned transformations:

$$h_1 = SiLU(\ W_1\ tanh(z) + b_1\ ) \tag{1}$$

$$h_2 = SiLU(\ LayerNorm(\ W_2\ h_1 + b_2\ )) \tag{2}$$

$$f = W_3\ h_2 + b_3 \tag{3}$$

For tensor i with element count n_i, the reconstructed weight vector is:

$$\hat{w}_i = tile(\ f,\ n_i\ ) \cdot \alpha_i + \beta_i \tag{4}$$

where $\alpha_i \in \mathbb{R}$ and $\beta_i \in \mathbb{R}$ are learned per-tensor scale and bias parameters, and tile(·, n) repeats the 256-dimensional feature vector to fill n elements. The full parameter set is $\varphi = \{W_1, b_1, W_2, b_2, W_3, b_3, \{\alpha_i\}, \{\beta_i\}, z\}$.

3.3 Training Objective

The hypernetwork is trained to minimize mean squared error across all tensors:

$$\mathscr{L} = (1/|\mathscr{B}|)\ \Sigma_{i \in \mathscr{B}}\ MSE(\ \hat{w}_i\ ,\ w_i\ ) \tag{5}$$

where $\mathscr{B}$ is a random mini-batch of tensor indices, and w_i are the dequantized target weights. Training uses AdamW with learning rate 10^{-3}, cosine annealing over 200 epochs, and gradient clipping at norm 1.0.

3.4 Dequantization

Q4_K_M tensors are dequantized block-wise (block size 256) using the embedded scale factors in the GGUF format. Q6_K tensors use block size 256 with 6-bit quantization. The dequantized float32 tensors serve as training targets.

3.5 The On-Chain Seed

After training, the learned latent z is extracted as:

$$z\_chain = tanh(z) \in [-1,\ 1]^{256} \tag{6}$$

serialized as 256 float32 values (1,024 bytes), and encoded as a hex string for on-chain storage. The seed hash H_seed = SHA256(z_chain) and model hash H_model = SHA256(model file) are stored alongside the seed for integrity verification.

Algorithm 1 — POKEBALL Reconstruction Protocol

```
Require: On-chain seed z_chain, public hypernetwork architecture H_φ

Fetch z_chain from smart contract via get_latent_code_hex()
Decode hex string to float32 array z ∈ R^256
Initialize H_φ with public architecture specification
Compute f = trunk(z) through shared layers
for each tensor i in model specification do
    Ŵᵢ ← tile(f, nᵢ) · αᵢ + βᵢ
    Reshape Ŵᵢ to target tensor shape sᵢ
end for
Verify: SHA256(reconstructed model) ≈ H_model
return reconstructed weight tensors {Ŵᵢ}
```

4 ON-CHAIN DEPLOYMENT

4.1 Smart Contract

The POKEBALL contract is implemented in ink! 4.3.0, compiled with rustc 1.85.0 and cargo-contract 3.2.0, and deployed on Aleph Zero mainnet. The contract ABI exposes the following read-only message selectors:

Method	Selector	Return	Description
`get_protocol()`	`0x42663fb5`	`String`	Protocol identifier
`get_model()`	`0x845c9da0`	`String`	Model name
`get_model_hash()`	`0xe4158aef`	`String`	SHA256 of model file
`get_seed_size_bytes()`	`0x8f1094fc`	`u32`	Seed size in bytes
`get_latent_code_hex()`	`0x517573f5`	`String`	Hex-encoded latent seed
`get_seed_hash()`	`0x2fba7019`	`String`	SHA256 of seed bytes
`get_compression()`	`0xd76be70b`	`String`	Compression ratio string
`get_all_metadata()`	`0x135c32ae`	`Tuple`	All fields at once

Table 1: Smart Contract ABI — POKEBALL (ink! 4.3.0)

4.2 Contract Address and Verification

Network	`Aleph Zero Mainnet`

Contract Address	5CXjoRcejjFrNsQuQw3BgM4qJ2aACYh2aA9wwqN5oTjjq2Kn
Code Hash	0x3925cad04533443cb829129e259dc53a2540a2aba094759fb4eda4796c33bce9
Language	ink! 4.3.0
Compiler	rustc 1.85.0
Build Mode	Release (wasm-opt Z passes)
Deployed	April 19, 2026
Explorer	https://alephzero.org/ecosystem/subscan

4.3 On-Chain Data Record

```
{
  "protocol": "POKEBALL",
  "model": "tinyllama-1.1b-chat-v1.0.Q4_K_M",
  "model_hash": "9fecc3b3cd76bba89d504f29b616eedf7da85b96540e490ca5824d3f7d2776a0",
  "seed_size_bytes": 1024,
  "latent_code_hex": "cb213cbb82823ebb2c41c0b85d170939c58337ba134f78bc...
   ...448a6b39caa13e3c787eddb9dda41d3a60bea6b981e56fb9c4e356bbda9b06bbc4b9...
   ...113a12e8653bf27477baf1318d3c449c4e3a1d79923cd1ba34bcb7a8acba0f6f17b...
   ...b34d945bb3783e338c2fdc93a75970fbbbf237dba533cc2b970aa4ebb864ab8bbae...
   ...83003a2316a43a8436f03afbbf99bbe19570ba56f1b3bad7b620ba140a0d3a3e849...
   ...4bcc1ab3d3baf10303a626490b99f424bee0cba5320003c0e7b28bb27cc31bb87c...
   ...d6138a8db9d3be312063c41b706ba62415ebab1d492ba0c4899bbefcc563b6139f9...
   ...ba9262e239",
  "seed_hash": "c209e0df2cb93215d6009108ab35c412a0dc3f7e2bbdc09b0fa2a75b1b6979e3",
  "compression": "654,296x"
}
```

Listing 1: POKEBALL On-Chain Seed Record (stored in contract state)

4.4 Smart Contract Source

```rust
#![cfg_attr(not(feature = "std"), no_std, no_main)]
use ink::prelude::*;

#[ink::contract]
pub mod m {
    use ink::prelude::string::String;
    #[ink(storage)]
    pub struct M {
        protocol: String,
        model: String,
        model_hash: String,
        seed_size_bytes: u32,
        latent_code_hex: String,
        seed_hash: String,
        compression: String,
    }
    impl M {
        #[ink(constructor)]
        pub fn new() -> Self {
            Self {
                protocol: String::from("POKEBALL"),
                model: String::from("tinyllama-1.1b-chat-v1.0.Q4_K_M"),
                model_hash: String::from("9fecc3b3..."),
                seed_size_bytes: 1024,
                latent_code_hex: String::from("cb213cbb..."),
                seed_hash: String::from("c209e0df..."),
                compression: String::from("654,296x"),
            }
        }
        #[ink(message)]
        pub fn get_latent_code_hex(&self) -> String {
            self.latent_code_hex.clone()
        }
        #[ink(message)]
        pub fn get_all_metadata(&self)
            -> (String, String, String, u32, String, String, String)
        {
            (self.protocol.clone(), self.model.clone(),
             self.model_hash.clone(), self.seed_size_bytes,
             self.latent_code_hex.clone(), self.seed_hash.clone(),
             self.compression.clone())
        }
    }
}
```

Listing 2: POKEBALL ink! Smart Contract (lib.rs)

5 COMPRESSION ANALYSIS

Property	Value
Model file size	637 MB (GGUF, Q4_K_M)
Number of tensors	363
Number of layers	22 transformer blocks

Hidden dimension	2048
Parameters	1.1B
Seed dimensionality	256 float32 values
Seed size (bytes)	1,024
Seed size (hex chars)	2,048
Compression ratio	**654,296×**
Hypernetwork params	~1.3M
Training epochs	200
Optimizer	AdamW, = 10^3

Table 2: Compression Statistics — TinyLlama-1.1B Q4_K_M

The compression ratio ρ is defined as:

$$\rho = |model\ file| \,/\, |z\_chain| = (637 \times 10^6\ bytes) \,/\, (1024\ bytes) \approx 654{,}296 \tag{7}$$

6 DISCUSSION

6.1 Novelty

To the best of our knowledge, this is the first deployment of a reconstructable LLM weight representation in a smart contract. The key distinction from prior work is that the on-chain data is not merely a fingerprint but a generative seed that allows for approximate model reconstruction without centralized dependencies.

6.2 Limitations

The reconstruction is lossy. The degree of approximation depends on the hypernetwork capacity. For this prototype, reconstruction fidelity is a proof of concept; it demonstrates the protocol but may not yield full inference quality matching the original weights.

6.3 Future Directions

Future work includes larger hypernetworks for higher fidelity, applying hierarchical latent codes to 7B+ parameter models, and implementing zero-knowledge proofs for weight integrity.

7 CONCLUSION

We have demonstrated POKEBALL: the first protocol for storing a reconstructable LLM representation in a smart contract. A 1.1B parameter model is compressed to a 1,024-byte latent seed and deployed to Aleph Zero mainnet. This establishes a new primitive for AI model permanence, ensuring that model identity and capability

can live forever on a decentralized ledger, protected from deprecation or deletion.

ACKNOWLEDGMENTS

We thank Matthias Christandl (QMATH, University of Copenhagen / Simons Institute for the Theory of Computing, Berkeley) for early conversations on hyperedge representations and hypergraph structure that informed this work.

REFERENCES

[1] Ha, D., et al. (2016). HyperNetworks. arXiv:1609.09106.

[2] Dao, J., et al. (2023). Bittensor Whitepaper.

[3] Ritual (2024). Decentralized AI Infrastructure.

[4] Xenova (2023). Transformers.js.

[5] Christandl, M., & Mitchison, G. (2012). Comm. Math. Phys., 261(3).

[6] Cardinal Cryptography (2021). Aleph Zero Ecosystem.