Neural-Policy Chess Inference Inside the EVM: A GenesisL1 Case Study and a General Template for On-Chain Scientific Machine Learning

Decentralized Science Labs

December, 2025

Abstract

Running neural networks *inside* an EVM smart contract has long been treated as impractical: dense arithmetic is costly, floating point is absent, memory is constrained, and determinism is mandatory. In this work we present an existence proof that these barriers are not fundamental. We describe a complete on-chain inference system for chess based on a compact policy network deployed as Solidity code and executed by the EVM, producing deterministic move logits and top-K selections under a legal-move mask. The deployment is performed on GenesisL1, an EVM-compatible blockchain whose compute-friendly execution parameters make such inference feasible; under typical "mainnet-like" EVM conditions, the same inference budget would be prohibitively expensive or fail to fit within practical call constraints.

The model is trained on approximately 2×10^7 positions from 2200+ Elo games, reaching about 25% validation top-10 accuracy when predicting the expert move index in the full fixed action space without providing legality as a network input. We formalize the state encoding, the 4672-action indexing scheme, the mask algebra, and the deterministic selection primitives implemented on-chain. Beyond chess, we argue that this result has broad significance: it demonstrates that *small, quantized neural networks can be decentralized and executed directly in smart contracts*, enabling verifiable scientific inference for bioinformatics, genomics, chemistry, physics, and astronomy, and supporting "small chat-like" question answering via constrained language models that map user prompts to structured, auditable outputs linked to committed datasets.

1 Introduction: Why This Was Considered Impossible

The EVM was designed for general-purpose verification and state transitions, not for neural inference. Conventional wisdom states that neural networks are an off-chain concern: the arithmetic intensity of matrix multiplications, the need for nonlinear activations, and the sheer parameter volume conflict with the EVM cost model.

The contribution of this work is not merely that we can *call* a model from a user interface. Rather, we show that the model itself can be encoded and evaluated *on-chain*, with deterministic outputs that are reproducible by any observer and suitable for protocol-level use. The achievement is an engineering and mathematical demonstration: inference is reduced to integer/fixed-point primitives, the action space is made finite and maskable, and selection is implemented with bounded loops.

A key enabling factor is the choice of deployment environment: GenesisL1 provides an EVM execution context configured to be compute-friendly (relative to typical mainnet deployments). In this setting, the core question becomes: *what minimal neural primitive can be made practical inside the EVM, and what does that enable?*

2 System Overview (On-Chain First)

The system is centered on a Solidity contract that implements a neural policy function. Conceptually, for a state s it produces logits $z(s) \in \mathbb{R}^{4672}$ over a fixed move index set $\mathcal{A}$, and supports masked selection over the legal subset $\mathcal{L}(s) \subseteq \mathcal{A}$.

The on-chain API exposes deterministic inference primitives:

```
weightsReady() -> bool

inferBest(state, legalMask)
  -> (bestIdx, bestLogit)

inferTopK(state, legalMask, K)
  -> (idx[0..K-1], logits[0..K-1])
```

The contract therefore acts as an on-chain "policy oracle" with two essential properties:

1. **Determinism:** outputs are a pure function of inputs.

2. **Verifiability:** any party can re-run the same call and obtain the same outputs.

An off-chain client may assemble inputs (state and legal mask) and apply optional search logic, but the scientific and engineering novelty resides in the on-chain mathematics: quantized inference and selection under a 4672-bit legality constraint.

3 State Encoding

We encode each chess position as a tuple consisting of piece bitboards and auxiliary state:

$$s = (\texttt{bb}[0..11],\ \texttt{stm},\ \texttt{castling},\ \texttt{epFile}).$$

- `bb`[i] is a 64-bit bitboard for the i-th piece plane (6 piece types per color, 12 total).
- `stm`$\in \{0,1\} indicates side-to-move (0 = White, 1 = Black)$.
- `castling`$\in \{0,\dots,15\} is a 4-bit mask for KQkq rights$.
- `epFile`$\in \{-1,0,\dots,7\} stores the en-passant file (or -1 if none)$.

Square indexing is fixed with $a1 = 0$ and $h8 = 63$, i.e.

$$\texttt{sqIdx}(f, r) = 8(r-1) + f,$$

where $f \in \{0,\dots,7\}$ is the file index and $r \in \{1,\dots,8\}$ is the rank.

This encoding is compact, EVM-friendly, and supports fast bitwise manipulation.

4 Action Encoding: 4672 Indices as a Maskable Algebra

A central engineering constraint is a *fixed* output dimension. We define a move index set $\mathcal{A}$ of size 4672 as:

$$\texttt{actionIdx} = 73 \cdot \texttt{fromSqIdx} + \texttt{planeIdx},$$

with `fromSqIdx` $\in \{0,\dots,63\}$ and `planeIdx` $\in \{0,\dots,72\}$, hence $|\mathcal{A}| = 64 \cdot 73 = 4672$.

The plane index encodes:

- Sliding moves: 8 directions times 7 distances (56 planes).
- Knight moves: 8 planes.
- Underpromotions: 3 promoted piece types (N, B, R) times 3 directions (forward, capture-left, capture-right) (9 planes).

This construction is fundamental: it transforms chess (a variable-branching game) into a fixed-index classification problem, enabling contract inference to produce a constant-size output and enabling legality filtering via a fixed-size bitmask.

5 Legal Mask Encoding (4672 Bits = 584 Bytes)

Legality is represented as a bitmask $m \in \{0,1\}^{4672}$, packed into 584 bytes. Let $m_i = 1$ indicate that action i is legal.

Given a bitmask in bytes `maskBytes`, legality is tested by:

$$m_i = (\texttt{maskBytes}[i/8] \gg (i \bmod 8)) \ \& \ 1.$$

This is on-chain friendly: it uses only indexing, shifts, and bitwise AND.

5.1 Why This Matters

The legal mask is the bridge between exact chess rules and approximate neural priors:

- The contract never needs to implement full chess legality.
- The model never needs legality as an input feature.
- Correctness is enforced by mask algebra at selection time.

6 Training Data and Objective (Policy Without Legal Features)

The network is trained on approximately 2×10^7 positions derived from 2200+ Elo games. Each sample is a pair $(s, a^\star)$ where $a^\star \in \mathcal{A}$ is the move played by the expert.

The learning objective is cross-entropy over the full action space:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(s,a^\star)} \log \left(\frac{\exp(z_{a^\star}(s))}{\sum_{a \in \mathcal{A}} \exp(z_a(s))} \right).$$

A key scientific feature of this setup is that the model is not given the legal mask as an input. Nevertheless, evaluation reports approximately 25% validation top-10 accuracy for predicting $a^\star$ within the top 10 logits in the fixed 4672-dimensional space. Interpreted literally, this implies that the network learns substantial structure of legal move geometry and chess semantics purely from data, even before explicit masking is applied at inference.

7 On-Chain Neural Inference: Integer Mathematics Under Determinism

The EVM has no floating point arithmetic. Practical on-chain inference therefore requires integer or fixed-point computation. The standard approach is:

1. Quantize weights and (optionally) activations.
2. Compute affine transforms with integer dot products.
3. Apply inexpensive nonlinearities (often piecewise-linear).
4. Carry scaling factors explicitly to keep values bounded.

Let x denote an input vector derived from the state encoding. A quantized affine layer can be written as:

$$y_j = \left\lfloor \frac{1}{S} \left(\sum_i w_{ji} x_i + b_j \right) \right\rfloor,$$

where w_{ji} and b_j are integers and S is a scaling constant. Accumulators typically require wider integer types to prevent overflow (e.g., 32-bit or 64-bit). The mathematical design task is to choose quantization scales so that:

- overflows are avoided,
- precision is sufficient for stable ranking,
- the computational footprint is compatible with EVM constraints.

7.1 Selection Primitives Implemented On-Chain

The contract must implement `inferBest` and `inferTopK`. A direct sort over 4672 elements is expensive. A common EVM-compatible approach is a bounded "streaming top-K" algorithm:

1. Initialize arrays `bestIdx[0..K-1]` and `bestLogit[0..K-1]`.
2. For each action i:
   - If illegal (mask bit is 0), skip.
   - Compute logit z_i by running the network for that action (or for the full head and indexing).
   - Insert into the top-K buffer if z_i exceeds the current minimum.

This yields time complexity $O(|\mathcal{A}|K)$ with small K (e.g., $K \leq 32$), which is far more realistic than a full sort for EVM execution.

7.2 GenesisL1 as an Enabler

The practicality of these loops is chain-dependent. GenesisL1, as used in this project, is configured to be compute-friendly for EVM workloads, enabling substantially larger view-call compute budgets than typical mainnet settings. In more restrictive EVM environments, an equivalently sized on-chain neural evaluation plus top-K selection can be impractical due to gas limits, call constraints, and cost structure. This distinction is part of the scientific contribution: it identifies a class of EVM environments where on-chain neural inference transitions from "theoretically definable" to "operationally feasible."

8 Why the Achievement Is Significant

The significance is not that a client can play chess. The significance is a new engineering fact:

> *A nontrivial learned function (neural policy inference) can be executed deterministically inside the EVM, with outputs that are directly usable by other smart contracts and verifiable by any observer.*

This is an existence proof for a larger category of decentralized scientific computation:

- compact neural surrogates can be placed on-chain as public utilities,
- inference can be made censorship-resistant and auditable,
- protocol rules can depend on model outputs without trusting a centralized server.

9 Beyond Chess: On-Chain Neural Models for Science

Chess is a convenient stress test because it requires combinatorial structure, legality constraints, and tactical sharpness. However, the underlying methods generalize.

9.1 Bioinformatics and Genomics

Many tasks in genomics reduce to compact classification/regression functions:

- sequence motif classification,
- variant effect scoring (small local windows),
- quality control and anomaly detection on summarized features.

With careful feature engineering (or small learned embeddings) and aggressive quantization, these functions can be realized as small neural networks whose inference is plausible under compute-friendly EVM conditions. On-chain inference enables:

- transparent scoring rules for shared datasets,
- verifiable eligibility decisions in decentralized data markets,
- auditability of scientific pipelines where a model output triggers a protocol event.

9.2 Chemistry, Physics, and Astronomy

Similarly, compact surrogate models appear in:

- molecular property prediction from descriptors,
- calibration functions and anomaly detectors in physics experiments,
- event filtering or prioritization in astronomy based on feature vectors.

The on-chain setting is especially compelling when a community needs a *single canonical scoring function* that cannot be silently changed. An on-chain model provides a fixed, globally consistent computation.

9.3 "Small Chat-Like" Language Models for Structured Q&A

Large language models are far beyond typical EVM budgets. However, a meaningful class of language-like models *is* compatible with on-chain principles if the scope is constrained:

- small-vocabulary intent classifiers,
- template-based generators with learned routing,
- small transformers or recurrent models with tiny context windows and heavy quantization,
- models that map text prompts to structured actions (tags, indices, schema fields).

In scientific settings, an important problem is *orientation in uploaded data*: given a dataset committed by hash (stored off-chain, referenced on-chain), users want consistent, auditable mappings from natural-language queries to dataset indices, fields, or analysis templates. A small on-chain language-like model can serve as a deterministic query router:

`prompt` $\mapsto$ `(datasetId, tableId, columnId, queryTemplateId)`.

The contract can then emit or store the structured output, making downstream interpretation reproducible. The model does not need to generate long prose; it needs to produce canonical structured decisions that a community can verify and build upon.

10 Determinism, Auditability, and Protocol Integration

On-chain inference has properties that typical ML deployments lack:

- **Replayability:** the same call yields the same output.
- **Public verifiability:** any node can recompute results.
- **Composability:** other contracts can depend on outputs.
- **Governance transparency:** upgrades (if allowed) are explicit.

For scientific applications, these properties enable a new design pattern:

Use a small on-chain model as a canonical scoring or routing primitive, and attach it to a larger off-chain data substrate via commitments.

This aligns with the constraints of blockchains (limited storage, costly compute) while still realizing the core value: decentralized, auditable inference.

11 Conclusion

We presented a GenesisL1 case study demonstrating that small neural networks can run directly inside an EVM smart contract and provide deterministic inference primitives suitable for decentralized use. In chess, this is realized as a policy oracle over a fixed 4672-action space with legality enforced by a 584-byte mask. The broader scientific conclusion is that on-chain neural inference is not a curiosity; it is an enabling technology for verifiable computation in many domains, from genomics to chemistry, and for constrained language-like models that map queries to structured, auditable outputs over committed datasets.

The long-term impact is architectural: when inference becomes a public, deterministic primitive, machine learning can become a component of protocol design rather than an external service. GenesisL1-like compute-friendly EVM environments make this feasible today for compact models, and the methods described here provide a template for future decentralized scientific ML systems.