AIVM: Proof of AI

The Settlement Primitive for Verifiable Machine Computation on Lux

Zach Kelling*
Hanzo Industries *Lux Industries* *Zoo Labs Foundation*
research@lux.network

June 2026

Abstract

We present the AIVM (AI Virtual Machine) and its settlement primitive, *Proof of AI* (PoAI): a protocol for committing, challenging, and settling deterministic integer AI computation. A versioned kernel profile fixes the consensus-visible operator semantics while a canonical transcript binds the job, model, input, graph, output, metering, provider, and destination. The arithmetic core uses Freivalds' matrix-product verifier over the prime field $\mathbb{F}_p$, $p = 2^{61} - 1$, on the exact-integer accumulator of a quantized `int8` matrix multiplication. Honest CPU and GPU implementations therefore agree byte-for-byte. For one committed matrix product, one unpredictable challenge has information-theoretic error at most $1/p \approx 2^{-61}$; two independent vectors reduce the local error to at most 2^{-122}. This local bound is not, by itself, a whole-model bound: a graph protocol must additionally ensure coverage of every operation, or quantify random sampling and guarantee transcript data availability. The current implementation provides an optimistic commitment-and-fraud-proof backend plus unit-tested typed-graph components; production integration into a complete model pipeline remains work in progress.

The A-Chain runs the sole Proof-of-AI useful-work consensus. It admits a demand-backed job before mining, fixes the model and execution policy, verifies that the committed computation was actually performed, consumes a global work nullifier, and finalizes one canonical receipt. Z-Chain is the P3Q proof-compression and settlement rollup: it batches A-Chain receipt transitions into a Plonky3-derived, pairing-free STARK/FRI settlement proof. Lux Quasar orders and finalizes the A-Chain and Z-Chain state on the post-quantum base layer. A miner may choose a payout destination, but neither Z-Chain nor that destination independently admits raw work or decides that computation occurred. Other chains verify the Z-settled, PQ-finalized A-Chain receipt and release the authorized asset representation. This gives the network one issuance schedule and no-double-mint ledger while allowing permissionless CPU or GPU execution and omnichain payout.

Contents

*zach@lux.network

1 Introduction

A blockchain settles claims about value because anyone can cheaply verify them. Bitcoin settles "this work was done" by making the work (a partial hash preimage) trivial to check and expensive to produce; the work itself is otherwise useless. The open question for a decentralized AI network is whether one can settle a far more valuable claim — "this AI computation was performed correctly" — with the same asymmetry: cheap to verify, expensive to fake, and *the work is the product*.

The AIVM answers yes for a deliberately constrained execution domain. Its settlement primitive, Proof of AI, makes a single observation precise: most work in a quantized neural-network pass is matrix multiplication $C = A \cdot B$, and a matrix product is exactly the object Freivalds' 1977 algorithm verifies probabilistically more cheaply than recomputation [1]. If the proof is anchored on the *exact integer* accumulator of a canonical quantized model — not a floating-point shadow that disagrees across hardware — the local check becomes bit-exact and deterministic. The protocol then adds transcript binding, graph chaining, data availability, challenge coverage, and A-Chain settlement. Only after those layers succeed may one unit of accepted computation authorize subsidy once, under the A-Chain's global nullifier state.

1.1 Contributions

1. **A common trace for supported inference, embedding, and training graphs (§4).** We define a typed deterministic-integer trace whose well-formedness plus verification of every operation imply, by induction, that the committed output equals execution on the committed input. This statement covers graphs expressible in the specified operation set; it is not a claim about arbitrary floating-point training stacks.

2. **An information-theoretic local soundness core (§2).** Freivalds over $\mathbb{F}_p$, $p = 2^{61} - 1$, on an exact-integer accumulator, with error $1/p$ per independent challenge vector and zero false-reject for an honest matrix product. Hash commitments and signatures remain computational post-quantum assumptions of the complete protocol.

3. **Cross-language byte parity (§5).** Rust and Go golden vectors share one field and challenge derivation, while a Solidity test verifies a Go-produced opening unchanged. This establishes wire compatibility of the tested fixtures, not production integration of every model kernel.

4. **A-Chain PoAI useful-work consensus (§6–§7).** A-Chain owns job admission, proof finality, the global work-nullifier set, and issuance authorization. Z-Chain rolls finalized receipts into P3Q settlement proofs; Lux provides post-quantum ordering and finality beneath both state machines. Destination chains consume certified receipts and do not run an independent PoAI acceptance path.

5. **Bitcoin-shaped economics applied to accepted useful work (§8).** A capped, halving, deflationary issuance schedule whose subsidy is authorized only by finalized A-Chain work receipts. Demand establishes usefulness; the execution proof establishes computational correctness.

1.2 The shape of the system

The reference contracts live at `luxfi/standard` under `contracts/ai/`, while the arithmetic core is mirrored in `hanzo-engine`, `luxfi/crypto`, and `ComputeWitnessLib.sol`. These components are at different maturity levels. The arithmetic, serialization, and small-graph tests are implemented; a production-scale proof emitted by a complete Zen model and consumed by A-Chain consensus is not yet the default inference path. The contracts are therefore reference mechanisms

and test surfaces until A-Chain integration, data availability, challenge finality, and deployment wiring are complete.

Mining-authority invariant. "Mine into another chain" means that an A-Chain job names a destination for settlement. It never means that Z-Chain or the destination chain runs PoAI, owns an independent mining spent set, or decides whether work occurred. A-Chain PoAI consensus finalizes exactly one issuance authorization; Z-Chain compresses and settles that authorization, and a destination adapter verifies it and releases the corresponding representation.

Three-layer finality and settlement. PoAI is the A-Chain domain consensus that decides whether an admitted useful-compute claim is valid and final. Z-Chain is the P3Q settlement rollup that batches and proves finalized A-Chain receipt transitions; it is not a second work-admission or mining authority. Quasar supplies validator ordering, fork choice, and finality for the A-Chain and Z-Chain state, with activated post-quantum authentication securing the exported settlement root. Thus "finalized by PoAI consensus," "settled through P3Q," and "secured by Lux consensus" describe complementary layers, not competing leader-election mechanisms.

1.3 Threat model

We assume compute providers may be Byzantine: they may return fabricated outputs, claim outputs they never computed, splice unrelated activations into a trace, withhold transcript data, replay one piece of work to many destinations, claim hardware they do not possess, or collude to extract subsidy without performing the work. We assume KECCAK-256 is collision- and preimage-resistant, ML-DSA is unforgeable, and A-Chain finality is not captured. The Freivalds equality test for one opened matrix product has information-theoretic error; the complete protocol does not, because commitment binding, authentication, data availability, challenge coverage, and consensus rely on additional assumptions. We require an unpredictable post-commitment public beacon and enough transcript availability for an honest challenger to reconstruct an inclusion proof before finality. Confidential TEE evidence is optional and has an explicit vendor trust assumption.

2 The Soundness Core: Freivalds over a Prime Field

2.1 Why a matrix product is the right primitive

Let a proof-bearing layer compute $C = A \cdot B$ with $A \in \mathbb{Z}^{t\times k}$, $B \in \mathbb{Z}^{k\times n}$, $C \in \mathbb{Z}^{t\times n}$. Recomputing C to check it costs $O(t \cdot k \cdot n)$ — exactly the work the prover did, so verification would be no cheaper than redoing the inference. Freivalds' insight is to check the product against a random vector $r \in \mathbb{F}_p^n$:

$$A \cdot (B \cdot r) \stackrel{?}{=} C \cdot r, \tag{1}$$

which costs $O(tk + kn + tn)$ — three matrix-vector products, an order of magnitude cheaper than the matrix-matrix product. This asymmetry is the whole point: the model is enormous, the witness is one opened matmul, and the verifier does a fraction of the prover's work.

2.2 The field and the exact-integer accumulator

We evaluate (1) over the Mersenne prime field $\mathbb{F}_p$ with

$$p = 2^{61} - 1. \tag{2}$$

The choice is deliberate. Each operand is an `int8` weight or activation reduced into $[0, p)$; the field is 61 bits, so a single 128-bit multiply of two reduced elements never overflows and a

modular reduction by a Mersenne prime is cheap. In the on-chain mirror, p is small enough that Solidity's `mulmod`/`addmod` stay exact under it.

Crucially, the operands are taken from the *exact integer* accumulator of the engine's quantized matmul — an $i8 \times i8 \to i64$ whole-K accumulation with no cross-block floating-point rescale. The Rust prover documents and implements this as the proof-bearing GEMM (`poi.rs`, `exact_matmul`):

Listing 1: The exact-integer matmul; identical in Go (`crypto/poi.ExactMatmul`).

```rust
pub fn exact_matmul(a: &Mat, b: &Mat) -> Mat {
    let (t, k, n) = (a.rows, a.cols, b.cols);
    let mut c = vec![0i64; t * n];
    for i in 0..t { for j in 0..n {
        let mut acc: i64 = 0;
        for x in 0..k { acc += a.data[i*k + x] * b.data[x*n + j]; }
        c[i*n + j] = acc;
    }}
    Mat::new(t, n, c)
}
```

Property 1 (Cross-backend determinism)**.** *For operand and dimension bounds that prevent accumulator overflow, integer matrix multiplication incurs no rounding and all valid reduction orders produce the same value over $\mathbb{Z}$. Alternatively, an execution specification may define identical modular or saturating overflow semantics. Under one of those explicit policies, honest CPU and GPU implementations produce the bit-identical accumulator C.*

This is the property the entire scheme stands on. A floating-point matmul may produce platform-dependent low bits, and a tolerance window complicates soundness. The production execution specification must therefore bind accumulator width, maximum inner dimension, overflow behavior, and dequantization rules rather than relying on the language's default integer or floating-point behavior.

2.3 Soundness

Theorem 1 (Freivalds soundness over $\mathbb{F}_p$)**.** *Fix A, B, C over $\mathbb{F}_p$ with $C \neq A \cdot B$. For r drawn uniformly from $\mathbb{F}_p^n$,*

$$\Pr_r[\, A(Br) = Cr \,] \;\leq\; \frac{1}{p}.$$

Proof. Let $D = C - A \cdot B \neq 0$ over $\mathbb{F}_p$. Then $A(Br) = Cr \iff Dr = 0$. Pick an entry $D_{ij} \neq 0$. Row i of Dr is the linear form $L(r) = \sum_\ell D_{i\ell}\, r_\ell$, which is not identically zero because $D_{ij} \neq 0$. Fix all coordinates of r except r_j; then L is a degree-one nonzero polynomial in the single variable r_j over the field $\mathbb{F}_p$, so it has at most one root. Hence $\Pr[L(r) = 0] \leq 1/p$, and $\Pr[Dr = 0] \leq \Pr[(Dr)_i = 0] \leq 1/p$. (This is the Schwartz–Zippel lemma [2, 3] specialized to degree one.) □

Corollary 1 (Two challenges)**.** *With k independent challenge vectors, a fabricated C passes all of them with probability at most $(1/p)^k$. For $p = 2^{61} - 1$ and $k = 2$,*

$$\left(\tfrac{1}{p}\right)^2 \;<\; 2^{-122}.$$

The Go verifier states and enforces exactly this bound (`crypto/poi/freivalds.go`): "Soundness error is $1/p \sim 2^{-61}$ per challenge vector; $k = 2$ vectors give $\leq 2^{-122}$."

Theorem 2 (Post-quantum local equality test)**.** *The soundness of Theorem 1 and Corollary 1, conditioned on an unpredictable challenge and a binding opening, is information-theoretic: it is a counting argument over a finite field and invokes no computational hardness assumption. A computationally unbounded adversary gains no advantage against that equality test.*

This is the structural reason the arithmetic core is post-quantum. The complete PoAI protocol additionally uses KECCAK commitments, ML-DSA authentication, A-Chain consensus, and an availability/challenge game. Those are computational or system assumptions and must not be collapsed into the local information-theoretic bound.

Theorem 3 (Zero false-reject)**.** *If $C = A \cdot B$ exactly over $\mathbb{Z}$ and every entry of A, B, C is reduced into $[0, p)$ consistently, then (1) holds for* every *r. Thus an honest prover is never rejected.*

Proof. Field reduction is a ring homomorphism $\mathbb{Z} \to \mathbb{F}_p$; if $C = A \cdot B$ over $\mathbb{Z}$ then the identity holds after reduction, so $Dr = 0$ for all r. By Property 1 the honest integer accumulator is reproducible, so the operands the verifier reduces are exactly the prover's. □

Theorem 3 is what makes the scheme safe to put on the critical path of an economic system: an honest miner cannot be slashed by bad luck, and a challenger who submits a "fraud proof" against honest work simply finds the check passes.

2.4 Challenge unpredictability

A prover must not be able to pre-fit (A, B, C) to a known r. The canonical Rust and Go derivation hashes a public, post-commitment beacon with the task and transcript root:

$$r[j][i] \;=\; \mathrm{BE\_u64}\big(\textsc{keccak}(\,\mathrm{seed} \,\|\, \mathrm{BE32}(j) \,\|\, \mathrm{BE64}(i)\,)[\,0{:}8\,]\big) \bmod p,$$

with $\mathrm{seed} = \textsc{keccak}(\mathrm{beacon} \,\|\, \mathrm{task} \,\|\, \mathrm{root})$. Because the beacon is fixed only *after* the prover commits the root, the challenge is unpredictable at commit time; because the derivation is a pure function of public inputs, both sides reproduce it without interaction.

2.5 From local checks to protocol soundness

Freivalds answers "is this opened matrix product correct?" It does not answer "did the verifier inspect the fraudulent operation?" Suppose a trace contains L operations, $b \geq 1$ of which are invalid, and the protocol samples s operations independently and uniformly. The probability that sampling misses every invalid operation is

$$\left(1 - \frac{b}{L}\right)^s.$$

If each selected matrix product is checked with k independent Freivalds vectors, a conservative acceptance bound is

$$\Pr[\text{accept invalid trace}] \;\leq\; \left(1 - \frac{b}{L}\right)^s + \frac{s}{p^k}.$$

For a sparse one-operation fraud in a large graph, the sampling term dominates; quoting only $1/p^k$ would be incorrect. A protocol may instead use an optimistic fraud-proof game in which independent watchers re-execute the graph and may challenge *any* invalid committed operation. In that design, security requires the complete transcript (or enough erasure-coded data to reconstruct it) to be available throughout the challenge window, plus a finality rule that prevents irreversible issuance before the challenge resolves.

The Go and Rust verifiers support multiple challenge vectors. The current Solidity witness derives one vector, so its local error is $1/p \approx 2^{-61}$, not 2^{-122}. Production parameters must state both s and k and bind them in the A-Chain execution policy.

2.6 What the core proves, and what it does not

The Freivalds core proves a *single opened* matmul output is genuine, conditioned on the commitment and challenge assumptions. Three gaps remain. First, a prover could commit correct matmuls on unrelated inputs; §4 addresses this with graph chaining. Second, a prover could return a different value than it committed; §3.4 binds the returned value to the accumulator. Third, a verifier may never inspect the bad operation; §2.5 makes that coverage term explicit. None of these checks proves that an answer is semantically true, useful, unbiased, or worth a particular reward. Demand-backed A-Chain job admission supplies usefulness; PoAI supplies execution accountability.

2.7 Verified test evidence for the core

The Rust core (`poi.rs`) and the Go core (`crypto/poi`) each ship a soundness suite that we ran to completion:

Test	Asserts	Lang
Honest product	$C = A \cdot B$ passes every challenge	Rust/Go
Tampered output	one flipped entry of C is rejected	Rust/Go
Signed reduction	negative int8 values reduce exactly	Rust
Sparse fabrication	one off-by-one entry in 16×16 is caught	Rust/Go
Challenge determinism	both sides derive identical r	Rust
Shape mismatch	invalid dimensions fail closed	Rust/Go
Golden parity	Rust r equals Go r byte-for-byte	Rust/Go

Table 1: Representative soundness-core tests (all passing).

At the recorded snapshot, the Go suite reported 22 passing tests and the Rust suite reported 27. A small Go fixture measured 1.13 ms per opening, or 882 verifications/second on one test-host thread. Section 9 states the limits of this measurement.

3 The Transcript: Committing a Forward Pass

A single matmul check is useless unless it is bound to a specific, committed computation. The transcript layer (`poi_transcript.rs`; Go mirror `crypto/poi/transcript.go`) makes "no valid compute proof, no mint" enforceable.

3.1 Deterministic kernel profile and canonical transcript

The consensus object is not an arbitrary GPU execution log. Each A-Chain job names a versioned *deterministic kernel profile*: the admitted operator set, tensor layout, integer widths, accumulator bounds, overflow behavior, reduction order, fixed-point scales, rounding rule, tie-breaks, and canonical byte encoding. A miner may implement that profile on a CPU, GPU, or another accelerator, but an optimization is admissible only when it produces the same profile bytes. Hardware speed changes who can execute a job economically; it does not change the computation A-Chain judges.

Execution emits one *canonical transcript*. Its header binds the A-Chain job and policy epoch, profile version, model and runtime commitments, input root, declared graph, output root, metering summary, miner, destination, and nonce. Its ordered body binds every typed operation to its input and output commitments. The Merkle root of that serialization is the portable receipt commitment consumed by the challenge game and, after PoAI finality, by P3Q

settlement. This distinction matters: deterministic kernels define the function; the canonical transcript proves which invocation of that function was claimed.

CPU/GPU "parity" is therefore a release conformance test, not a second consensus mechanism. Golden vectors must show that every supported backend produces byte-identical kernel outputs, leaves, transcript roots, challenges, and openings. A backend that fails a vector is excluded from the profile; consensus never accepts a tolerance band to make the backend fit.

3.2 Commitment

Each proof-bearing matmul is committed as a leaf binding its three exact-integer operands under a domain tag:

$$\text{leaf} \;=\; \textsc{keccak}\big(\texttt{"hanzo/poi/matmul-leaf/v1"} \,\|\, \text{mat}(A) \,\|\, \text{mat}(B) \,\|\, \text{mat}(C)\big),$$

where $\text{mat}(M) = \text{BE32}(\text{rows}) \,\|\, \text{BE32}(\text{cols}) \,\|\, (\text{each } M_{ij} \text{ as BE64})$ is the canonical serialization. The leaves are folded into a binary KECCAK-Merkle root using the index-fold convention (even index $\to$ KECCAK(node $\|$ sibling), odd $\to$ KECCAK(sibling $\|$ node)); odd levels duplicate the last node. This is the *transcript root* the prover posts on-chain.

3.3 Challenge and opening

A challenger derives the index of the matmul to open from the beacon and root,

$$\text{index} \;=\; \textsc{keccak}(\text{beacon} \,\|\, \text{root})[\,0{:}8\,] \bmod \text{length},$$

unpredictable before the prover commits the root. The prover *opens* that index, revealing (A, B, C) and the Merkle inclusion path. Verification is the conjunction of binding and correctness (`verify_opening`):

1. **Binding (Merkle inclusion).** The revealed operands hash to the committed leaf at `index` under `root`. A prover who swaps in different operands fails here.
2. **Correctness (Freivalds).** $C = A \cdot B$ under k beacon-bound challenge vectors derived from KECCAK(beacon $\|$ root $\|$ BE64(index)). A fabricated output fails here.

Lemma 1 (Opening soundness)**.** *For an opening that is Merkle-included under `root`, an honest $C = A \cdot B$ passes every challenge. If $C \neq A \cdot B$, it passes k independent unpredictable Freivalds vectors with probability at most $1/p^k$. The Go and Rust paths commonly test $k = 2$; the current Solidity path uses $k = 1$.*

This lemma is the local interface used by the off-chain watcher and on-chain fraud predicate. It says nothing about the chance that an invalid operation is opened; that is the separate coverage and availability problem of §2.5.

3.4 Sound forward pass: the returned value is the committed value

The transcript binds operands, but an honest-looking transcript is worthless if the layer *returns* something other than what it committed. `ProvableLinear` (`poi_forward.rs`) closes this. It computes $y = \text{dequant}(\text{quant}(x) \cdot W_{i8})$ on the exact-integer accumulator, commits the operands ($A = x_{i8}$, $B = W_{i8}$, $C = x_{i8} \cdot W_{i8}$), and returns the dequantization of the *committed* C:

$$y[t][o] \;=\; C[t][o] \cdot \text{xscale}[t] \cdot \text{wscale}[o],$$

where the scales are public model parameters. Dequantization is therefore a public affine map of C.

Theorem 4 (Output-commitment binding)**.** *Committing C and the canonical scale encoding commits the intended dequantized y. Exact consensus replay additionally requires a fixed-point dequantization and rounding specification; ordinary floating-point evaluation with a tolerance is not byte-exact across all backends.*

The current output-binding test checks returned y against dequant(committed C) with floating-point tolerance. That is useful engine evidence but is not a consensus determinism proof. A production proof-bearing output must remain integer or fixed-point through the committed decision boundary, or define a canonical bit-exact dequantizer. A separate SwiGLU fixture exercises three committed matmuls.

4 Graph Composition for Deterministic Programs

4.1 The gap the graph closes

Per-matmul soundness still permits a prover to commit individually-correct matmuls computed on activations it never derived from the prompt — splicing an unrelated tensor mid-stream. The graph layer (`poi_graph.rs`) makes the *wiring* of the computation part of the commitment.

Definition 1 (Computation trace)**.** *A `GraphTrace` is a sequence of typed operations over hash-identified activations. An activation tensor M is named by*

$$\mathrm{act}(M) = \mathit{KECCAK}(\texttt{"hanzo/poi/activation/v1"} \,\|\, \mathrm{mat}(M)).$$

Each operation commits a leaf binding its kind, input commitments, inline parameters, and output commitment. The trace also declares its input activations.

The prototype operation set contains the following pure deterministic integer functions. It is sufficient for the included attention, MoE, and linear-SGD fixtures; production model support requires a complete manifest for every operator actually executed by the chosen architecture and optimizer.

Op	Semantics (exact integer)
`MatMul`	$C = A \cdot B$ — the only Freivalds-checked op
`Add`	residual out $= x + y$
`ReLU`	out $= \max(0, x)$
`Scale`	out $= (x \cdot \mathrm{num})/\mathrm{den}$ (e.g. attention's $1/\sqrt{d}$)
`TopKRoute`	MoE router: top-k indices per row, lowest-index tie-break
`Gather`	expert dispatch: out$[i] = x[\mathrm{idx}[i]]$
`Softmax`	deterministic fixed-point (Q16) integer softmax
`Transpose`	out$[j][i] = x[i][j]$ — the one op a backward pass adds

The softmax is the one subtlety: a floating-point softmax is not reproducible across backends, so the trace uses a pure integer scheme (max-shift for stability, an I-BERT-style fixed-point exponential [4], and an integer normalize) whose output is bit-identical for prover and challenger. The test `test_int_softmax_is_a_valid_distribution` confirms each row sums to $\approx 2^{16}$, is monotone in the logit, and is bit-reproducible.

4.2 The two checks

Verification of a trace composes two checks:

1. **Well-formed (chaining).** Every operation's input commitments reference either a declared trace input or the output commitment of an *earlier* operation. A break — an op reading an activation no prior op produced — is visible from the committed leaves alone, with no openings. `verify_chaining` returns false on the first dangling reference.

2. **Locally correct (per-op).** An opened operation recomputes its output from its inputs — a `MatMul` by Freivalds under a beacon-bound challenge, a glue op (`Add`, `ReLU`, `Scale`, `TopKRoute`, `Gather`, `Softmax`, `Transpose`) by direct recomputation — and the recomputed result's commitment equals the committed output.

4.3 Global soundness

Theorem 5 (Whole-graph soundness)**.** *If a trace is well-formed and every operation is locally correct, then the committed final output equals the deterministic forward pass applied to the committed input.*

Proof. Induction over the topological order of the trace (the operations are recorded in execution order, which is a valid topological order). *Base:* a declared input activation's committed value is, by definition, the true input. *Step:* consider operation o with inputs $u_1, \ldots, u_m$. By chaining, each u_ℓ is either a declared input or the output of an earlier operation; by the inductive hypothesis its committed value is the true intermediate value. Local correctness of o guarantees that the committed output of o equals the operation applied to those true inputs. Since each op is the exact-integer deterministic function the reference forward pass uses, the committed output of o is the true intermediate value at o. Applying the step to the final operation yields the claim. □

Because everything is exact integer, activation commitments and glue recomputation are reproducible. The fixtures express attention's $QK^\top$ and AV products plus an MoE route, gather, expert, and combine sequence. The tests verify every operation in those standalone blocks, including integer softmax.

Theorem 5 is conditional: its antecedent says *every* operation is locally correct. A verifier that checks only a sampled subset has not established that antecedent; its guarantee is instead the coverage bound of §2.5. The current tests exercise standalone blocks and synthetic graphs, not a complete production Zen inference with transcript emission on the normal serving path.

4.4 Training is a graph, not a separate proof system

For the included linear-SGD fixture, the additional operation is `Transpose`: gradients are $\mathrm{d}A = \mathrm{d}C \cdot B^\top$ and $\mathrm{d}B = A^\top \cdot \mathrm{d}C$. The same `MatMul` kind and Freivalds check cover its forward and backward products. Real training systems additionally require optimizer state, normalization, loss functions, stochasticity policy, precision rules, distributed collectives, and overflow behavior to be specified as deterministic trace operations before they fall within the theorem.

Corollary 2 (One proof framework for admitted graphs)**.** *Inference, embedding, and training computations expressible entirely in the admitted deterministic operation set are instances of Theorem 5. A fabricated opened gradient matmul is subject to the same local check as a fabricated opened inference matmul.*

This is verified for the fixture: `test_training_step_is_just_a_graph` traces a linear-layer SGD step, checks the graph, verifies every op, and fabricates a weight gradient that Freivalds catches. An input commitment identifies declared data; it does not prove authorship, licensing, completeness, or real-world provenance. Those require separate evidence and policy.

4.5 Adversarial test evidence

The graph module ships a battery of attack tests, each asserting a specific cheat is caught:

Across the four `poi` source modules, `cargo test -lib poi` runs **27 tests, 0 failed** (Property 1 through Corollary 2 are exercised by name in this set).

Attack test	What it proves is caught
Chain splice	operation reads an unproduced activation
Output substitution	opening claims a different output
Fabricated matmul	committed matrix output is invalid
Forged softmax	attention weights are altered
Forged route	token is sent to a different expert

Table 2: Graph adversarial fixtures at the recorded implementation snapshot.

5 Cross-Implementation Conformance

A proof is only useful at settlement if the prover, the off-chain watcher, and the chain agree on every byte. PoAI is mirrored three ways and the mirrors are tested against each other, not merely asserted to match. These parity suites are engineering gates for an execution-profile release; they are not a user-facing proof tier and do not weaken the requirement for A-Chain PoAI finality.

5.1 One hash, one field, one serialization

All three implementations use Ethereum KECCAK-256, the field $p = 2^{61} - 1$, shared domain tags, canonical matrix serialization, the same Merkle fold, and the same challenge-index encoding. The Solidity arithmetic is pure and portable, although only an A-Chain-finalized receipt may authorize mining payout.

5.2 The on-chain witness

`ComputeWitnessLib.sol` is the on-chain proof-of-inference witness check. Its core is Freivalds in pure Solidity, using `mulmod`/`addmod` under p so the field arithmetic is exact:

Listing 2: Freivalds over $\mathbb{F}_p$ on-chain (`ComputeWitnessLib.sol`).

```
function freivalds(Matrix memory a, Matrix memory b, Matrix memory c,
    uint256[] memory r)
     internal pure returns (bool)
{
    if (a.cols != b.rows || b.cols != c.cols || a.rows != c.rows || r.
        length != b.cols) return false;
    uint256[] memory br  = _matvec(b, r);   // length k
    uint256[] memory abr = _matvec(a, br);  // length t
    uint256[] memory cr  = _matvec(c, r);   // length t
    for (uint256 i = 0; i < abr.length; i++) if (abr[i] != cr[i])
        return false;
    return true;
}
```

The library exposes two complementary predicates. `verifyOpening` returns true iff the opening is Merkle-included and Freivalds-valid. `provesFraud` returns true iff it is Merkle-included and Freivalds-invalid. The current Solidity implementation derives one challenge vector and accepts dense matrices with each dimension capped at `MAX_DIM` = 1024. That cap prevents unbounded allocation but does not make a worst-case opening inexpensive: calldata and hashing still include every element of A, B, and C. The repository describes bounded slicing, but the current whole-matrix leaf format does not yet provide a production slice-to-parent binding. Production deployment therefore requires either a committed tiling scheme, an A-Chain native verifier, or a succinct proof backend, together with measured gas and payload bounds.

5.3 Proven parity

Parity of the tested primitives is enforced by golden vectors shared by the engine, Go verifier, and Solidity witness suite. The Solidity suite checks:

- `test_LeafParityWithProver` — the Solidity `matmulLeaf` equals the prover's leaf;
- `test_GoOpeningVerifiesOnChain` — an opening produced by the Go verifier verifies inside the EVM unchanged;
- `test_FabricatedOutputCaughtOnChain` and `test_SwappedRevealCaughtOnChain` — the two attack classes are caught on-chain;
- `test_FreivaldsOverField` — the field arithmetic matches.

All five pass in the recorded suite. The strongest demonstrated end-to-end fixture is Go opening $\rightarrow$ Solidity verification; Rust $\leftrightarrow$ Go parity is established through shared golden vectors. A complete Rust model-run transcript consumed by A-Chain remains an integration milestone.

6 The AIVM as the Lux Settlement Layer

PoAI consensus establishes that useful work was actually done: usefulness comes from a pre-admitted, demand-backed job whose requested output and consumer are fixed before mining, while execution evidence binds the model, input, trace, metering, and output. The A-Chain is the only state machine allowed to decide that this mining claim is valid. It owns the job, usefulness policy, execution policy, challenge beacon, proof status, work nullifier, reward calculation, and final receipt. Z-Chain compresses and settles finalized receipt transitions through P3Q, and Quasar orders and finalizes both state machines. Other chains consume this finality; neither Z-Chain nor a destination is an alternate PoAI authority.

6.1 Mining a proof into a coin

An A-Chain mining job binds a task identifier, accepted model and runtime, input commitment, proof policy, reward budget, expiration, destination chain, destination recipient, and nonce. The miner commits an execution transcript before the challenge beacon is known. A-Chain then performs, or accepts under its configured finality policy, the following checks:

1. **Usefulness.** The job existed before the miner's result, was admitted under a demand or network-service policy, binds a real consumer and deliverable, and cannot be created and self-certified solely by the miner. This is a protocol definition of useful work, not a proof that an answer is universally true or socially valuable.
2. **Binding.** The result binds the A-Chain job, model, runtime, input, output, miner, and destination selected before execution.
3. **Execution evidence.** The configured backend verifies deterministic recomputation, an optimistic transcript after its challenge period, a succinct proof, or an approved attestation policy. A bare signature or output quorum may settle a paid subjective task but cannot authorize subsidy by itself.
4. **PoAI finality.** Optimistic evidence is not final while a fraud window is open. No irreversible issuance receipt may be exported until PoAI consensus resolves every challenge and transcript-availability obligation.
5. **Global uniqueness.** A-Chain consumes the job's canonical work nullifier atomically with finalization.

6. **Receipt.** A-Chain emits one receipt containing the reward authorization and destination. The destination may release only that amount to only that recipient.

The reference proof binding collapses the task, intent, model, prompt, open block, operator, output, and runtime into one `reportData`. In the canonical topology, that binding is judged and finalized on A-Chain. A Solidity deployment elsewhere is a receipt-verification or testing adapter, not a second mining authority. The current optimistic backend reports attestation while a challenge is pending, and the reference miner consumes that pending status. Production A-Chain issuance must instead require final backend state; slashing after an irreversible mint is not finality.

6.2 The global no-double-mint ledger

The global no-double-mint set is A-Chain state. Its key is derived from the A-Chain job/challenge identity and the accepted proof commitment, not from a destination-local chain id. Finalizing the job consumes that key exactly once. A second result for the same mining challenge cannot produce a second issuance authorization, regardless of requested destination.

A key based only on $(\text{model}, \text{prompt}, \text{output})$ is insufficient as the canonical job identity: two legitimate paid jobs may deterministically request the same computation, while an attacker may try to pre-claim a predictable tuple. The A-Chain task identifier, challenge nonce, policy epoch, and receipt identifier provide explicit domain and lifecycle binding. Destination replay protection is then simple: each destination records the finalized A-Chain receipt/nullifier it has consumed. That local consumed set prevents replay of *the same A-Chain authorization*; it does not decide whether work was valid.

6.3 Post-quantum-rooted omnichain settlement

The omnichain design uses Lux as the post-quantum root of useful-work truth:

1. A-Chain PoAI consensus admits the job, verifies that the bound work was performed, consumes the global nullifier, and commits the final receipt under its receipt root.

2. Z-Chain batches finalized A-Chain receipt transitions and, once the PoAI AIR and verifier are activated, verifies their Plonky3-derived P3Q STARK/FRI settlement proof. Z-Chain proves compression and settlement of A-Chain decisions; it does not admit mining jobs or validate an unrelated raw-work claim.

3. Quasar finalizes the A-Chain receipt root, the corresponding Z-Chain settlement root, and the validator set. The activated PQ authentication policy certifies the roots and heights; selecting ML-DSA alone is insufficient without validator-key and threshold wiring.

4. A native atomic boundary, Warp/Teleport message, or quorum-authenticated relay carries the Z-settled A-Chain receipt and inclusion proof to the destination named before mining.

5. The destination verifies base-layer finality, Z-Chain settlement, A-Chain receipt inclusion, destination binding, amount, recipient, and local non-consumption, then releases the authorized representation exactly once.

Before the P3Q PoAI circuit is activated, Z-Chain may record only an authenticated, already-final A-Chain receipt root; it MUST NOT represent that record as a succinct proof of the underlying work. Activation is fail-closed: the PoAI AIR, prover, verifier, recursion policy, and A-to-Z state-transition binding must be complete, audited, and enabled by protocol version before a

P3Q proof can replace the direct finality path. A shadow or replay phase may compare candidate proof outputs against finalized receipts, but it has no mint authority. "Staged activation" is deployment safety, not a weaker proof accepted by consensus.

`AChainRootOracle.sol` is the reference EVM verifier for the direct, pre-rollup path: a quorum of A-Chain validators signs the root and height with ML-DSA, and anyone may relay it. The production omnichain adapter SHOULD verify the Z-settled commitment or a native proof of A-Chain/Z-Chain finality. A destination MUST NOT accept a raw `ComputeProof`, TEE quote, Freivalds opening, or miner signature as a substitute for that PQ-authenticated receipt. This is how mining can settle into any supported chain while Proof-of-AI consensus exists only on A-Chain.

Property 2 (Authenticated A-Chain root)**.** *Forging an accepted root requires either breaking the configured authentication or violating A-Chain finality. This replaces a trusted single relayer with an authenticated finality proof; the precise security level depends on the deployed validator-set and threshold protocol, not merely on selecting ML-DSA as a signature algorithm.*

This is verified by the `AChainRootOracle.t.sol` suite (7 tests, passing), which exercises quorum acceptance, sub-threshold rejection, non-ascending-key rejection, stale-height rejection, and unregistered-validator rejection.

6.4 Bonded providers and automated slashing

`MinerStakeRegistry.sol` requires a miner to bond capital in proportion to the compute capacity it declares; the bond is skin in the game. Three orthogonal authorities, one per concern:

- the **miner** controls its own bond (bond / request-unbond / withdraw), with a cooldown so a discrepancy can be proven before it exits;
- the **slasher** — the compute-proof fraud verifier — slashes a bond on a *proven* discrepancy, with no vote, just the math (the on-chain consumer of `provesFraud`);
- **governance** (a Safe) may blacklist a miner for off-protocol harm the math cannot catch.

A miner is `eligible` to mine iff bonded above a floor, bonded in proportion to declared capacity, not exiting, and not blacklisted. Slashing works *during* the unbond cooldown, so a miner cannot dodge a fraud proof by exiting first.

Property 3 (Operator-safe)**.** *By Theorem 3, an honest miner's openings always verify, so `provesFraud` can never return true against honest work; the slasher therefore cannot slash an honest bond. Slashing is reachable only by a Merkle-included opening whose Freivalds check fails — a genuinely fabricated matmul.*

The recorded stake-registry suite covers bonding, the capacity floor, unbond cooldown, slasher authorization, and blacklist veto.

7 Governance: A Leaderless Quorum of Operator LLMs

The AIVM is governed not by a council of humans but by the network's own node operators reasoning with their language models, settling canonical decisions on-chain. The design is decomplected across contracts, each one concern.

7.1 Sortition: leaderless committee selection

`Sortition.sol` selects a committee by cryptographic sortition. An operator is in the committee for a round iff its seeded ticket falls below a threshold sized so that, in expectation, $\sim$`size` of `population` registered operators are selected:

$$\text{ticket} = \textsc{keccak}(\text{seed} \,\|\, \text{operator}), \qquad \text{selected} \iff \text{ticket} < \left\lfloor \frac{2^{256}}{\text{population}} \right\rfloor \cdot \text{size}.$$

Selection probability is size/population, so an adversary controlling fraction f of the population controls $\sim f$ of the committee. To control a *majority* of a committee it must control a majority of the *whole registered population*, not merely race to fill the first `size` slots. This is the equal-vote regime — one operator, one ticket, no stake weighting, so a sub-threshold whale cannot buy disproportionate seats. The seed must be fixed *after* operators commit and be unpredictable to them (a future block hash or VRF beacon); membership is one KECCAK per check, so no on-chain enumeration of the operator set is needed.

Property 4 (Leaderless)**.** *There is no sequencer, no privileged operator, and no leader. Committee membership is determined by post-commitment public randomness; the structural Sybil bar is a population majority, and the economic bar is a non-refundable registration cost.*

`Sortition.t.sol` verifies this with `test_ExpectedCommitteeSize`, `test_Deterministic`, and `test_CaptureRequiresPopulationMajority` (all passing).

7.2 Categorical decisions: AIGovernor

`AIGovernor.sol` produces a canonical YES/NO decision (with a confidence bucket) by aggregating signed, structured operator-LLM verdicts. Each verdict is authenticated by `ecrecover` over the exact canonical preimage, the recovered signer must equal the sender, signatures are low-S with $v \in \{27, 28\}$, and one verdict is admitted per operator per task. To defeat copying, a commit-reveal mode seals each operator's commit during a commit window and admits the reveal *strictly after* the window closes, so no operator can observe a peer's revealed fields before its own commit is sealed; the commit is operator-bound. `settle` has no access modifier — it is permissionless, gated only by the state machine and a deadline (so no one can front-run the threshold-th honest vote to suppress a forming quorum). The tally selects the largest group by consensus key; if it meets the threshold (a strict majority, $\geq n/2+1$), the decision settles. There is no leader anywhere in this path.

7.3 Continuous values: AIParams

Where the governor decides a categorical policy, `AIParams.sol` decides a continuous *value*: each operator's LLM proposes a number within a declared range, and the chain settles to the *median* of a quorum of proposals and makes it the live knob value. The median is deliberately unweighted — one operator, one proposal — and is Byzantine-robust to any minority below 50%. The committee is sortition-selected with a post-commitment seed (the open-block hash), the population is snapshotted at open, and `settle` is again permissionless. `AIParams.t.sol` (16 tests, passing) verifies the median computation, the in-range invariant, and the sortition gate.

7.4 From a settled decision to an arbitrary on-chain action

`AIExecute.sol` is the single surface validator consensus uses to read and act on-chain, decomplected by risk into three tiers, one mechanism:

- **Read.** Any getter is `staticcall`'d and handed back as a typed value (a bool, a number, an address, a word), so consensus can query arbitrary on-chain state structured.

- **Enact (Tier 1, no timelock).** A knob the validators have *decided* in `AIParams` is spliced as the sole argument of `target.selector(value)`; a 32-byte word ABI-encodes to any single type, so one call sets a bool / uint / int / address / bytes32. Low-risk and idempotent because the value *is* the consensus output.

- **Execute (Tier 2/3, windowed).** An *arbitrary* operation — any target, any method, any arguments, any native value — that the validators approved by its hash, gated by a timelock window, predecessor ordering, one-shot replay protection, a guardian veto, and an optional policy guard.

The operation hash binds `chainid` and the executor instance, so an approval can never be replayed on another chain or another executor.

The authority for execution is supplied by `AIApproval.sol`, an adapter (not a second governor) that binds a settled YES verdict to an operation through a single identity:

$$\text{a Thought's promptHash} \equiv \text{the AIExecute operation hash.}$$

So "should consensus execute operation X?" is literally the governance question the operators answered. An opener poses a Thought whose `promptHash` equals `hashOperation(op)`; the operators run their LLMs and settle; if the decision is SETTLED with vote YES, anyone may permissionlessly `confirm` it, which stamps the approval at the current block time — and *that* timestamp is where the timelock window begins. There is no admin, no owner, and no override: a NO, abstain, unsettled, or non-existent Thought yields no approval, so `execute` reverts. The only path to an approval is a real quorum YES. `AIApproval.t.sol` (7 tests, passing) verifies the settled-YES binding, the one-shot guard, and the rejection of every non-YES state.

7.5 Governance as a second proof of useful cognition

Consensus itself can produce useful cognitive artifacts, but a vote is approximately one bit and a quorum signature proves agreement rather than execution. A governance participant may earn demand fees for an accepted judgment. Subsidy requires a separate proof-bearing A-Chain job that binds the participant's model, prompt, and output. `AIConsensusMiner.sol` is a reference contract for this conjunction; in the canonical topology it cannot create an independent mint seam on C or another destination. A-Chain finalizes the proof and consumes the same global issuance allowance used by every other mining job. `AIGovernanceBridge.sol` may index the resulting Proof-of-Thought receipt, but the receipt becomes subsidy-bearing only through A-Chain finality.

8 Economics: Bitcoin-Shaped, Fair-Mined on Useful Work

8.1 The issuance schedule

The target AI monetary policy has two fixed pools:

- $D = 1{,}000{,}000{,}000{,}000$ AI is allocated at genesis to the Hanzo DAO treasury. It is not mineable and cannot be reclassified as mining emission.

- $M = 1{,}000{,}000{,}000{,}000$ AI is reserved exclusively for PoAI miner and validator rewards. It has no discretionary mint path: every unit must fit the schedule and a finalized A-Chain receipt.

- total nominal supply is bounded by $S_{\max} = D + M = 2{,}000{,}000{,}000{,}000$ AI before burns. Destination representations do not add supply.

The mining pool follows a Bitcoin-shaped geometric emission. Let H be the governance-fixed halving interval in finalized A-Chain blocks (nominally four years), $k = \lfloor (h - h_0)/H \rfloor$ the era, and $f \in [0, 1)$ progress through it. Within an era the aggregate allowance vests linearly and its slope halves at the next era:

$$\text{allowedMining}(h) \;=\; M(1 - 2^{-k}) \;+\; f \cdot M2^{-(k+1)}. \tag{3}$$

Thus the first era may emit at most 500 billion AI, the second 250 billion, the third 125 billion, and so on. Jobs share the available era budget according to finalized useful work under the active reward policy; raw self-reported FLOPs do not determine payout.

Theorem 6 (Capped, continuous, exact)**.** *allowedMining*$(h_0) = 0$*; the allowance is continuous and increasing; and allowedMining*$(h) \to M$*. The geometric sum of era budgets is exactly* M*:* $\sum_{k \geq 0} M2^{-(k+1)} = M$*. Consequently total nominal issuance never exceeds* $D + M = 2$ *trillion AI, and burns can only reduce it.*

The canonical settlement function clamps mining issuance to allowedMining(h) − mintedMining, so cumulative mining and validator rewards cannot exceed (3). Only the canonical ledger owns this counter. Destination contracts conserve against finalized A-Chain authorizations and canonical escrow; duplicating the counter and schedule on multiple EVMs would multiply supply and is forbidden. The current reference `AICoin.sol` still implements a legacy one-billion, no-preallocation schedule and must be upgraded and tested before this monetary policy can activate.

8.2 Deflation

`AICoin` is burnable, and the protocol burns a governed fraction of every settled fee. Net supply change per block is subsidy(t) − burn(t); since subsidy $\to 0$ while fee burn persists under use, net supply turns deflationary once the chain is in steady use.

8.3 Why this is the next Bitcoin

Bitcoin's social contract is a capped asset whose issuance is earned by publicly verifiable work. PoAI applies that economic shape to demand-backed AI execution while preserving the distinct roles of A-Chain work consensus, Z-Chain P3Q settlement, and Lux Quasar finality:

- **Capped with fair mining** (Theorem 6): two trillion total, a fixed one-trillion DAO allocation, and a separate one-trillion proof-earned pool with halving and a deflationary tail.
- **Public and permissionless** (§6.1, 6.2): anyone may perform an open A-Chain job under its admission and proof policy; the finalized receipt binds the reward recipient and optional destination.
- **Leaderless** (Property 4): no sequencer, no privileged operator; governance is sortition over the whole operator population.
- **Operator-safe** (Property 3): bond-and-slash, never exploitable against honest work, by the zero-false-reject theorem.
- **Post-quantum-oriented** (Theorem 2, Property 2): the local arithmetic check is information-theoretic, while commitments, authentication, and finality use explicitly stated post-quantum and consensus assumptions.

The intended difference is that the work is useful because an admitted job requests it. PoAI can establish execution accountability for supported deterministic graphs; it does not establish semantic truth or market value. A-Chain PoAI governs eligibility for AI-work rewards and subsidy, Z-Chain P3Q compresses and settles finalized receipt transitions, and Quasar secures and finalizes both chains.

8.4 Why existing AI clouds participate

PoAI is designed as an incremental revenue rail, not only a replacement marketplace. A centralized cloud, a decentralized provider, or an idle workstation may attach a job-specific transcript to inference, embedding, evaluation, or admitted training work it is already paid to run. The provider keeps its ordinary demand fee and, if PoAI consensus finalizes the evidence, earns a declining share of the mining allowance. The same adapter can accept open decentralized jobs when capacity would otherwise be idle.

This "compute once, earn twice" claim is valid only when proof capture is substantially cheaper than the original job and validators do not replay the full accelerator workload. Exact-integer commitments plus CPU-verifiable openings supply that asymmetry for supported operations: the provider performs an $O(n^3)$ matrix product, while a CPU checks a challenged product in $O(n^2)$ plus commitment overhead. Full-work accountability still requires graph binding, unpredictable post-commitment challenges, coverage or an activated succinct proof, transcript availability, and verifiable metering.

Subsidy eligibility must resist circular demand. A miner-created job, a self-reported GPU-hour, a copied output, or a customer signature alone cannot mint AI. The active policy must bind independent demand or a governed network-service budget, cap rewards per epoch, reject replay through the global nullifier, and make the reward no larger than the available schedule. These controls turn cloud workloads into public evidence without turning fake invoices into a mint.

9 Verified Test Evidence (Aggregate)

The following suites are implementation evidence for individual components. They are not evidence of a production end-to-end mining deployment.

Suite	Result	Command
`crypto/poi` (Go verifier)	22 passed, 0 failed	`go test ./...`
`hanzo-engine poi` (Rust core)	27 passed, 0 failed	`cargo test -lib poi`
`contracts/ai` (Solidity, 20 suites)	256 passed, 0 failed	`forge test`

Table 3: Recorded component-test results at the paper's implementation snapshot. Counts may change as suites evolve.

The recorded Go fixture measured 1.13 ms per small opening on its test host. This is not an on-chain gas benchmark or a production-model benchmark. Before launch the project still requires a real Zen model to emit a complete transcript on the normal execution path, a published availability protocol, measured worst-case challenge payloads, a finalized-not-pending issuance gate, and an end-to-end A-Chain receipt exported to and consumed by a destination chain.

10 Post-Quantum Primitives

The AIVM targets post-quantum security across its arithmetic, authentication, and transport layers. The following address assignments and candidate primitives are defined in the Lux pre-

compile namespace; registration does not by itself establish production deployment or threshold-protocol maturity:

Primitive	Address	Role
ML-KEM (FIPS 203)	`0x012201`	module-LWE KEM
ML-DSA (FIPS 204)	`0x012202`	module-LWE signature; the AChainRootOracle verifier
SLH-DSA (FIPS 205)	`0x012203`	hash-based signature
Pulsar	`0x012204`	module-LWE threshold research implementation
P3Q	`0x012205`	Plonky3-derived, pairing-free STARK/FRI settlement substrate
X-Wing KEM	`0x002221`	X25519 + ML-KEM-768 hybrid (encapsulate)

Table 4: Post-quantum precompiles (verified addresses from each precompile's `contract.go`).

The ML-DSA precompile is the intended verifier for authenticated A-Chain and Z-Chain root certificates. Production claims require node wiring, active validator keys, threshold policy, downgrade prevention, and deployment tests. P3Q supplies the Plonky3-derived, pairing-free STARK/FRI substrate intended for the Z-Chain settlement rollup, but the exact PoAI AIR, recursive batch policy, A-to-Z state binding, and end-to-end verifier remain work in progress until audited and activated. Pulsar remains threshold-signature research and must not be described as a deployed malicious-secure DKG/signing service until that protocol and its networking are complete.

Prompt confidentiality for proof-of-inference is provided by `luxfi/crypto/promptseal` — a pure-Go HPKE envelope (not a precompile) that seals a user's prompt to an operator's registered public key under the X-Wing hybrid KEM (X25519 + ML-KEM-768), HKDF-SHA256, and ChaCha20-Poly1305, with domain tag `"hanzo/poi/prompt-seal/v1"`. Hybrid means the seal stays confidential if *either* X25519 *or* ML-KEM-768 holds; only the chosen operator, inside its compute boundary, can open it. The matching on-chain surface is the X-Wing *encapsulate* precompile at `0x002221`; key generation and decapsulation are intentionally excluded on-chain because they would expose secret material, and run off-chain in the `promptseal` library.

11 Related Work

zkML. Zero-knowledge proofs of ML inference [8] can provide succinct non-interactive verification, usually with substantial prover overhead and system-specific computational assumptions. Pairing-based constructions are not post-quantum. The Freivalds core is cheaper for one opened matrix product, but the complete optimistic system additionally pays for commitments, availability, watcher re-execution, and dispute latency. A fair comparison must measure the complete protocol on the same model and threat policy.

Trusted execution environments. TEE attestation assumes hardware, firmware, collateral, measurement, and revocation trust. The AIVM accommodates TEE evidence for confidential jobs, but the intended public deterministic path is hardware-independent exact-integer execution. That path is not production-final until transcript availability and A-Chain challenge finality are integrated.

Optimistic rollups. PoAI's fraud-proof structure — commit a root, allow anyone to challenge an opening within a window, slash on a proven discrepancy — shares optimistic rollups' philosophy [9], specialized to matrix computation where the fraud proof is a single Freivalds check with $O(\log n)$ Merkle overhead rather than a full re-execution.

Proof of work. Bitcoin [5] couples work directly to leader election and consensus security. A-Chain PoAI is instead a useful-work validity consensus: it determines that an admitted job

was actually performed and may authorize issuance; Z-Chain P3Q compresses and settles that finalized state; Lux Quasar supplies base-layer ordering and finality. The computation is useful because a real, pre-admitted consumer job requests and consumes its result.

Matrix-multiplication proof of work. Komargodski and Weinstein [6] propose reusing arbitrary matrix multiplication for both an external workload and proof of work; BTX [7] explores a header-derived finite-field MatMul instantiation. Deterministic kernels, canonical transcripts, cross-device parity tests, and staged proof activation are useful engineering patterns. A chain-derived matrix race, however, does not by itself settle a requested model execution: unless the product is bound to a demand-backed job, its utility is hardware adjacency rather than a consumed AI result. Exact replay also makes verification the same asymptotic order as execution, while delegating replay to signed GPU attestors adds an explicit trust assumption. AIVM therefore adopts deterministic integer computation and proof-friendly commitments without making AI execution the leader-election oracle. A-Chain PoAI consensus validates useful work; Z-Chain P3Q settles the proof batches; Lux supplies the PQ-secured base layer; destination chains consume the resulting receipt.

12 Conclusion

Proof of AI is a useful-work consensus: it validates that a pre-admitted, demand-backed AI job was actually executed and produced its committed result. Its strongest implemented primitive is an exact-integer Freivalds check with zero false-reject for an honest opened matrix product and information-theoretic local error. Typed graph commitments, cross-language fixtures, evidence backends, and settlement contracts extend that primitive, but their composition requires explicit coverage, availability, finality, and deployment assumptions. The A-Chain is the sole PoAI authority that admits mining jobs, finalizes proofs, consumes work nullifiers, and authorizes subsidy. Z-Chain is the P3Q proof-compression and settlement rollup for those finalized transitions, and Lux Quasar plus activated PQ authentication make the A/Z settlement root the omnichain base-layer fact. Other chains verify that receipt and route value; they never create an independent proof or supply schedule. The remaining engineering milestone is an end-to-end production model run whose transcript is available, challenged, finalized by A-Chain PoAI consensus, proven and settled through the activated P3Q rollup on Z-Chain, PQ-certified by Lux, and consumed exactly once at a destination. Until that test is green, this paper is a protocol and implementation report, not a claim of a completed production mining network.

Acknowledgments

This work was supported by Hanzo Industries, Lux Industries, and the Zoo Labs Foundation. The canonical contract implementation lives at `luxfi/standard` under `contracts/ai/`; the soundness core lives in `hanzo-engine` (`src/poi*.rs`) and `luxfi/crypto` (`crypto/poi/`).

References

[1] R. Freivalds, *Probabilistic machines can use less running time*, Information Processing (IFIP Congress), 1977, pp. 839–842.

[2] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, Journal of the ACM 27(4), 1980, pp. 701–717.

[3] R. Zippel, *Probabilistic algorithms for sparse polynomials*, in Symbolic and Algebraic Computation (EUROSAM), 1979, pp. 216–226.

[4] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, K. Keutzer, *I-BERT: Integer-only BERT Quantization*, ICML, 2021.

[5] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

[6] I. Komargodski and O. Weinstein, *Proofs of Useful Work from Arbitrary Matrix Multiplication*, arXiv:2504.09971, 2025.

[7] BTX Contributors, *BTX Reference Node: MatMul Proof of Work*, https://github.com/btxchain/btx, accessed August 2026.

[8] D. Kang, T. Hashimoto, I. Stoica, Y. Sun, *Scaling up Trustless DNN Inference with Zero-Knowledge Proofs*, arXiv:2210.08674, 2022.

[9] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, E. W. Felten, *Arbitrum: Scalable, private smart contracts*, USENIX Security, 2018.

[10] National Institute of Standards and Technology, *FIPS 204: Module-Lattice-Based Digital Signature Standard (ML-DSA)*, 2024.

[11] National Institute of Standards and Technology, *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*, 2024.

[12] M. Barbosa, D. Connolly, et al., *X-Wing: The Hybrid KEM You've Been Looking For*, IACR ePrint 2024/039, 2024.