Hyperspace: A Peer-to-Peer Automated Intelligence Economy

Varun Mathur

varun@hyper.space

March 2026

ABSTRACT

We present Proof of Intelligence (PoI), a consensus mechanism for decentralized AI networks in which the "mining" is running AI experiments — training runs, inference tasks, research trials — and the network's collective intelligence is an emergent property of this process rather than any individual node's capability. Each node participates in a continuous experiment loop: execute a computation, prove that execution via a zero-knowledge proof over WASM (zkWASM, Groth16 on BN254), share results through a GossipSub-connected Research DAG, and adopt what works. Intelligence is not a node property to be measured; it is a network property that compounds as the loop runs at scale. This is how science works — no single experiment proves genius, but the method produces progress.

Benchmark challenges (ARC-AGI, SWE-bench, MATH) serve as onboarding: a fast capability check when a node first joins. Once a node has a contribution history in the Research DAG, its benchmark score becomes secondary to its experimental track record. Inference verification uses the Hidden State Commitment Protocol (HSCP), which proves a node ran a model locally by committing to internal hidden states that cannot be reconstructed from API outputs alone.

We demonstrate that PoI achieves Byzantine fault tolerance under standard assumptions ($f < n/3$), inherits economic security through staking and slashing, and introduces a novel security property we term the *Intelligence Opportunity Cost Bound*: the cost of attacking a PoI network scales with the opportunity cost of the intelligence required, making attacks irrational by a margin orders of magnitude larger than in hash-based or capital-based consensus. A complete implementation is deployed on the Hyperspace network, with zkWASM execution proofs committed on-chain via the AGENTCOMMIT (0xF3) opcode.

1. INTRODUCTION

1.1 The Experiment Loop as Consensus Work

Blockchain consensus mechanisms face a fundamental design tension: the "work" that secures the network should be (a) hard to fake, (b) easy to verify, and (c) ideally useful. Bitcoin's Proof of Work [1] satisfies (a) and (b) but fails (c) — SHA-256 hashes are immediately discarded after mining. Proof of Stake [2] replaces computational work with economic collateral, but introduces capital concentration dynamics and "nothing at stake" problems. Neither mechanism produces lasting value from the consensus work itself.

Proof of Intelligence begins from a different premise. The consensus work is AI experimentation — training runs, inference tasks, hyperparameter searches, architecture

trials — and the results persist. Each node on the network runs a continuous loop:

1. **Execute** an experiment (training step, inference batch, research trial)
2. **Prove** execution via zkWASM (Groth16 on BN254, O(1) on-chain verification)
3. **Share** results via GossipSub into a content-addressed Research DAG
4. **Adopt** what works — replicate successful experiments, build on others' results

This loop is the core mechanism. No individual node needs to demonstrate that it is "intelligent." Each node demonstrates that it participated in the experiment loop — that it ran real computation, proved it ran, and contributed results. Intelligence emerges as a network property: the Research DAG accumulates findings, successful experiments get replicated and extended, and the collective capability of the network grows over time. This is how science works. No single experiment proves genius, but the method — hypothesis, experiment, peer review, replication — produces progress.

The analogy to existing blockchains is precise. In Bitcoin, miners compete to find hash preimages; the work secures the chain but produces nothing else. In PoI, nodes compete to run useful experiments; the work secures the chain AND produces a growing corpus of AI research results. The "blocks" in the Research DAG are experiment commits that build on each other, and the "proof of work" is the zkWASM execution proof that the experiment actually ran.

1.2 Why Not Proof of Useful Work or zkML?

Several projects have proposed "Proof of Useful Work" (PoUW) where mining involves training ML models [3][4] or performing scientific computation [5]. These approaches face a fundamental problem: **training work is hard to verify cheaply**. Verifying that a model was trained correctly requires either re-training (which defeats the purpose) or trusted hardware attestation (which reintroduces centralization).

Zero-knowledge proofs for ML (zkML) [8] offer cryptographic verification that a specific model produced a specific output. But current zkML is limited to small models (<100M parameters), imposes 100-1000x overhead, and requires circuit recompilation for each model architecture.

PoI combines two verification strategies that avoid these limitations:

- **Proof of Execution** (zkWASM): "This computation ran as claimed" — the Agent Virtual Machine (AVM) compiles training steps and inference to WASM, executes them, and produces a Groth16 proof over the execution trace. WASM is deterministic by specification, sidestepping the GPU floating-point reproducibility problem. Verification is O(1) via a single pairing check on BN254. This is already implemented in the blockchain binary.
- **Proof of Intelligence** (benchmark challenges): "Given this challenge, my system produced this correct answer" — verification cost is O(1) (compare answer to ground truth). This serves as onboarding and periodic capability checks.
- **Proof of Inference** (HSCP): "I ran this model locally and can reveal its internal hidden states" — the Hidden State Commitment Protocol proves local execution by committing to intermediate layer activations that cannot be reconstructed from API outputs alone.

The key insight is that zkWASM, already present in the AVM, solves the experiment verification problem more directly than zkML. Rather than arithmetizing a neural network (expensive, architecture-specific), we arithmetize the WASM virtual machine itself

(general-purpose, architecture-agnostic). Any computation compilable to WASM — training, inference, data processing — is verifiable through the same circuit.

1.3 Contributions

1. **Loop-based consensus**: A consensus mechanism where the "mining" is running AI experiments in a continuous loop (execute, prove, share, adopt). Intelligence emerges as a network property from this process, not as a measured attribute of individual nodes.
2. **zkWASM experiment verification**: Execution proofs via a gnark zkWASM circuit (Groth16 on BN254) in the Agent Virtual Machine, committed on-chain via the AGENTCOMMIT (0xF3) opcode. O(1) verification. WASM determinism eliminates the GPU reproducibility problem.
3. **Hidden State Commitment Protocol (HSCP)**: A Merkle-tree-based protocol for verifying that inference was performed locally, by committing to hidden states that cannot be reconstructed from API outputs.
4. **Benchmark onboarding**: ARC-AGI, SWE-bench, MATH, and other benchmarks as capability checks for new nodes joining the network. Once a node has a contribution history, its benchmark score becomes secondary to its experimental track record.
5. **Security analysis**: Byzantine fault tolerance ($f < n/3$), incentive compatibility, and the *Intelligence Opportunity Cost Bound* — a novel property establishing that the cost of attacking a PoI network scales with the opportunity cost of the intelligence required.
6. **Reference implementation** deployed on the Hyperspace network, covering the full protocol stack from challenge generation through experiment verification.

---

2. RELATED WORK

2.1 Proof of Work

Bitcoin [1] introduced the concept of using computational puzzles (finding hash preimages below a target) to achieve distributed consensus. The security of PoW derives from the thermodynamic cost of computation: an attacker must expend more energy than all honest miners combined. This provides strong security guarantees but at enormous environmental cost (~150 TWh/year for Bitcoin as of 2025) with zero useful output.

The hashrate arms race has led to extreme centralization: as of 2025, three ASIC manufacturers (Bitmain, MicroBT, Canaan) produce >95% of mining hardware, and five mining pools control >65% of hashrate. This centralization was not anticipated in the original Bitcoin design and represents a failure mode where the "permissionless" ideal collides with economies of scale in specialized hardware.

2.2 Proof of Stake

Ethereum's transition to Proof of Stake [2] addressed PoW's energy waste by replacing computation with economic collateral. Validators lock tokens (32 ETH) and are slashed for misbehavior. This dramatically reduces energy consumption but introduces new concerns:

- **Capital concentration**: Wealthy entities can acquire disproportionate influence.

Liquid staking derivatives (Lido, Rocket Pool) further concentrate validation power.

- **Nothing at stake**: In naive PoS, validators can vote on multiple forks at no cost. Slashing conditions partially address this but add protocol complexity.
- **Long-range attacks**: An attacker who acquires old validator keys can create alternative histories. Checkpointing mitigates but doesn't eliminate this.
- **Minimum viable issuance**: The network must pay validators enough to incentivize staking but not so much as to cause excessive inflation. Finding this balance is an ongoing challenge.

2.3 Proof of Useful Work

Coin.AI [3] proposed using deep learning model training as the mining puzzle. The key insight — that ML training is hard to perform but relatively easy to verify (by testing the resulting model) — is sound in principle but faces practical challenges: training runs are long (hours to days), verification requires running the model on a test set (non-trivial compute), and the training objective must be standardized across all miners.

Proof-of-Learning [4] formalized this approach, defining the "learning proof" as a sequence of model checkpoints that demonstrates training progress. Verification involves checking that gradient updates are consistent with the claimed training data and hyperparameters.

Both approaches prove that *training happened*, not that *intelligence was produced*. A miner could train a model that converges on the training set but fails to generalize — still earning rewards for useless work.

More recently, Karpathy [26] described an "autoresearch" pattern in which AI systems autonomously run experiments, evaluate results, and iterate — observing that this "looks like a blockchain — commits that build on each other, proof of work is experimentation." Dai [27] asked the natural follow-up question: can you build proof-of-useful-work on top of autoresearch? PoI is a direct answer: the experiment loop IS the consensus work, zkWASM execution proofs replace hash puzzles, and the Research DAG replaces the linear blockchain with a content-addressed graph of experimental results.

2.4 Bittensor (TAO)

Bittensor [6] is the most prominent attempt at AI-based consensus. Miners register on "subnets," each managed by a subnet owner who defines the evaluation criteria. Validators score miners using the "Yuma consensus" — a mechanism that rewards miners whose outputs align with the validator majority.

Bittensor's design has several known failure modes:

1. **Centralized evaluation**: Subnet owners control the scoring rules, creating a trust bottleneck. The network's security reduces to trusting ~32 subnet owners, not trustless mathematics.
2. **Conformity incentive**: Yuma consensus rewards agreement with the majority, not correctness. Miners are incentivized to produce outputs that match what other miners produce, not outputs that are actually good. This rewards imitation over intelligence.
3. **Model identity gaming**: Miners claim to run specific models but face no cryptographic verification. Academic analysis has documented widespread model substitution (running cheaper models while claiming to run expensive ones) [7].
4. **Staker-biased emissions**: A significant portion of TAO emissions flow to delegators (stakers) rather than miners

who perform actual work, creating PoS-like capital concentration dynamics.

2.5 Verifiable Computation and zkML

Zero-knowledge proofs for machine learning (zkML) [8] enable cryptographic verification that a specific model produced a specific output. Projects like EZKL, Giza, and Modulus compile neural network architectures into ZK circuits.

Current limitations: - **Overhead**: 100-1000x computational overhead for proof generation - **Model size**: Practical only for small models (<100M parameters) as of 2025 - **Inflexibility**: Circuit must be compiled per model architecture; changing the model requires recompilation

zkML is complementary to PoI: it proves *which model* ran (Proof of Running), while PoI proves *how good the output is* (Proof of Intelligence). However, for experiment verification — the core PoI workload — zkWASM provides a more practical path. Rather than arithmetizing each neural network architecture into a dedicated ZK circuit, zkWASM arithmetizes the WASM virtual machine itself. Any computation that compiles to WASM (training steps, inference, data processing) is verifiable through a single general-purpose circuit. The Hyperspace AVM already contains a gnark zkWASM circuit (Groth16 on BN254) that serves this purpose, with proofs committed on-chain via the AGENTCOMMIT (0xF3) opcode.

2.6 AI Benchmarks as Intelligence Measures

François Chollet's seminal paper "On the Measure of Intelligence" [9] provides the theoretical foundation for our challenge design. Chollet defines intelligence as:

> "*The intelligence of a system is a measure of its skill-acquisition efficiency over a scope of tasks, with respect to priors, experience, and generalization difficulty.*"

This definition implies that intelligence benchmarks should: 1. Test novel tasks (not memorizable from training data) 2. Require genuine abstraction (not pattern matching) 3. Span diverse cognitive domains (not just one skill) 4. Have graduated difficulty (discriminate between capability levels)

The ARC-AGI benchmark [10] was designed to embody these principles. Each task is unique, requiring on-the-fly abstraction to solve. This makes ARC-AGI-style challenges ideal as PoI onboarding benchmarks: you cannot "cram" for them, and solving them requires genuine capability. Once a node has established a track record of experimental contributions, benchmark scores become secondary to contribution history — but they remain valuable as periodic capability checks and for difficulty calibration.

2.7 Autoresearch and Distributed Experimentation

Karpathy [26] described a pattern he called "autoresearch" — AI systems that autonomously propose hypotheses, run experiments, evaluate results, and iterate. He observed that the resulting workflow "looks like a blockchain — commits that build on each other, proof of work is experimentation." This observation is foundational to PoI's design.

Flywheel [28] introduced the Research DAG concept: a directed acyclic graph where each node is an experiment that references its predecessors, forming a content-addressed history of research progress. Unlike a linear chain, a DAG allows parallel lines of investigation that can merge when results are synthesized.

PrimeIntellect's INTELLECT/DiLoCo [29] demonstrated decentralized training at scale

using asynchronous gradient gossip, proving that meaningful ML work can be distributed across untrusted nodes without centralized orchestration. Their approach validates the technical feasibility of the experiment loop but does not provide cryptographic verification of the work performed.

PoI synthesizes these threads. The experiment loop (execute, prove, share, adopt) is Karpathy's autoresearch made into a consensus mechanism. The Research DAG is the data structure that accumulates results. zkWASM execution proofs provide the cryptographic verification that INTELLECT/DiLoCo lacks. Dai's question — can you build proof-of-useful-work on top of autoresearch? — has a concrete answer: yes, by proving execution via zkWASM and structuring results in a Research DAG with content-addressed experiment commits.

---

3. SYSTEM MODEL

3.1 Network Architecture and Node Hierarchy

The Hyperspace network consists of N nodes, each identified by an Ed25519 keypair. Nodes communicate via libp2p over WebRTC (browser) or TCP/QUIC (desktop). Coordination uses GossipSub pub/sub messaging and Kademlia DHT for content routing.

All nodes participate in the experiment loop (Section 3.6). The network defines four node types in ascending order of responsibility:

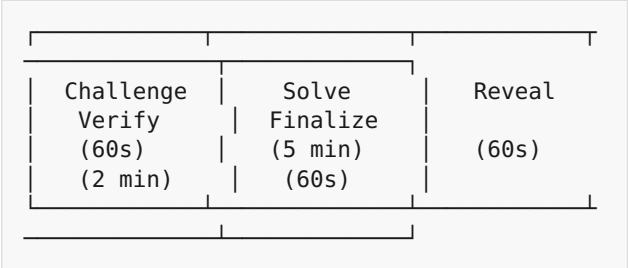
1. **Inference nodes**: Run inference for users. Participate in the experiment loop by executing assigned inference tasks, proving execution via zkWASM, and sharing results. No stake required.
2. **Routers** (Hydra DHT): Route requests across the network, maintain the DHT, and relay experiment results via GossipSub. Contribute to the Research DAG by aggregating and forwarding experiment commits from inference nodes.
3. **Full nodes**: Run experiments (training steps, hyperparameter searches, architecture trials) in addition to inference. Contribute original research results to the Research DAG. Require hardware attestation via Pulse.
4. **Validators**: Full nodes that additionally stake tokens (minimum 1,000) and participate in BFT consensus. Validators verify zkWASM execution proofs, vote on experiment validity, and finalize epochs. The validator set is permissionless and open — any node with sufficient stake and hardware (verified by Pulse WASM-based resource attestation) can register.

Every node type runs the same experiment loop. The distinction is in what additional responsibilities each tier carries: validators stake and verify, full nodes run original experiments, routers relay, and inference nodes execute assigned work. Intelligence is not localized to any tier — it emerges from the collective activity of all nodes contributing to the Research DAG.

3.2 Epoch Lifecycle

Time is divided into epochs of duration T (default: 10 minutes, matching Bitcoin's block time). The epoch is the verification cycle for the experiment loop — the cadence at which the network checks that nodes are doing real work, finalizes experiment results, and

distributes rewards. Each epoch proceeds through five phases:



Phase 1 — Challenge Distribution (60s): The epoch leader generates K challenges (default: 3) from a VRF seed and the most recent finalized block hash. For new nodes without a contribution history, these challenges serve as onboarding capability checks. For established nodes, they are periodic verification that capability has not degraded. Challenges are broadcast to all validators via GossipSub.

Phase 2 — Solve and Commit (5 min): Validators solve challenges using their local AI models. Concurrently, all nodes continue running experiment loop iterations (Section 3.6), submitting experiment commits to the Research DAG. For each challenge solution, the validator computes a commitment hash H(solution || nonce) and broadcasts the commitment. The commitment is binding (prevents changing the answer later) and hiding (reveals nothing about the solution).

Phase 3 — Reveal (60s): After the commitment deadline, validators reveal their solutions and nonces. Anyone can verify that the revealed solution matches the earlier commitment.

Phase 4 — Verification (2 min): A VRF-selected committee of sqrt(N) validators independently verifies each proof. Committee members check execution traces (including zkWASM proofs for experiment commits), timing proofs, model attestations, and grade challenge solutions against ground truth. The committee votes to APPROVE or REJECT each proof, requiring 2/3 agreement.

Phase 5 — Finalization (60s): Rewards are calculated and distributed based on both challenge performance and experiment contributions. Slashing is applied for misbehavior. Finalized experiment commits in the Research DAG become permanent. The epoch transitions to the next.

3.3 Challenge Generation (Onboarding and Periodic Verification)

Benchmark challenges are primarily an onboarding mechanism. When a new node joins the network, it has no contribution history in the Research DAG. Challenges provide a fast capability check: can this node solve reasoning, coding, and mathematical problems at a level that makes it a useful participant in the experiment loop? Once a node has a track record of verified experiment commits, its benchmark score becomes secondary — the contribution history is the proof.

Challenges also serve as periodic capability checks for established nodes, ensuring that hardware and model quality have not degraded. The challenge mechanism is not the core consensus work; the experiment loop (Section 3.6) is.

Challenges must be: - **Deterministic**: All validators derive the same challenges from the same seed - **Unpredictable**: No validator can predict challenges before the epoch begins - **Diverse**: Challenges span multiple cognitive domains - **Calibrated**: Difficulty adapts to maintain a target solve rate

We achieve these properties through the following construction:

Input: VRF output σ from the epoch leader (verifiable but unpredictable), block hash h of the most recent finalized block (unknown until finalization).

Seed: `seed = SHA256(σ || h || epoch_number)`

PRNG: A deterministic xorshift128 PRNG initialized from the seed generates all random choices for the epoch (challenge types, sub-types, parameters, problem instances).

Category selection: Challenges are drawn from six categories with configurable weights:

Category	Weight	Sub-Types	Verification Method
Reasoning	25%	Syllogistic, arithmetic, spatial, temporal, counterfactual	Exact match
Coding	25%	Implementation, bug-fix, optimization, test generation	Code execution
Classification	15%	Intent, sentiment, topic, NER	Exact match
Retrieval	15%	Semantic search, QA, fact verification	Ranking (nDCG)
Multi-step	15%	Decomposition, tool-use, planning, proofs	Step-by-step
Creative	5%	Open-ended generation	Rubric grading

Each challenge includes ground truth, enabling deterministic automated verification without human judgment (except the 5%-weighted creative category, which uses rubric-based heuristic scoring).

Difficulty calibration: An exponential moving average ($\alpha = 0.3$) of recent solve rates adjusts difficulty to target 60% solve rate at MEDIUM difficulty. If solve rates exceed the target, difficulty increases; if they fall below, difficulty decreases. This is analogous to Bitcoin's difficulty adjustment targeting 10-minute block times.

Onboarding vs. steady state: For a node's first 100 epochs, challenge performance is the primary determinant of its reward share and reputation. After 100 epochs with a verified contribution history, the weighting shifts: 30% challenge performance, 70% experiment contribution quality (measured by adoption rate in the Research DAG — how often other nodes build on this node's results).

3.4 Proof Structure

Each validator's proof bundles four components:

1. **Execution Trace** — A Merkle tree of all AI operations performed during solving (model inference calls, tool invocations, reasoning steps, code execution). The Merkle root commits to the full trace, but only K entries (default: 8) are disclosed for efficiency. Selective disclosure reduces on-chain proof size to ≤10 KB while maintaining verifiability.
2. **Timing Proof** — A Verifiable Delay Function (VDF) output proving the computation took at least T seconds. We use an iterated hash construction (with plans to adopt Wesolowski VDFs [11] for O(log n) verification). Hardware calibration during validator registration normalizes timing across different hardware.
3. **Model Attestation** — Cryptographic binding between the solution and the claimed model. Four trust levels:
4. Self-reported (0.4 confidence)
5. Output fingerprint matching (0.7 confidence) — statistical properties of model output (token entropy, vocabulary distribution, response length distribution)
6. Pre-committed model hash (0.9 confidence) — weight hash committed before seeing challenge

7. TEE attestation (1.0 confidence) — hardware-attested execution environment
8. **Solution Signature** — Ed25519 signature over the solution, binding it to the validator's identity.

3.5 Committee Verification

Inspired by Algorand's cryptographic sortition [12], we use VRF-based random committee selection:

Committee size: `min(50, max(7, floor(sqrt(N))))` where N is the validator count.

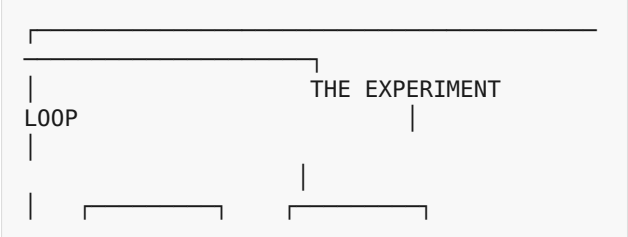
Selection: Each validator computes `score = SHA256(vrfSeed || challengeId || epoch || validatorId)`. Validators with scores below the threshold (calibrated for target committee size) are selected. Selection is weighted by the validator's historical intelligence score (0.5 + score × 0.5).

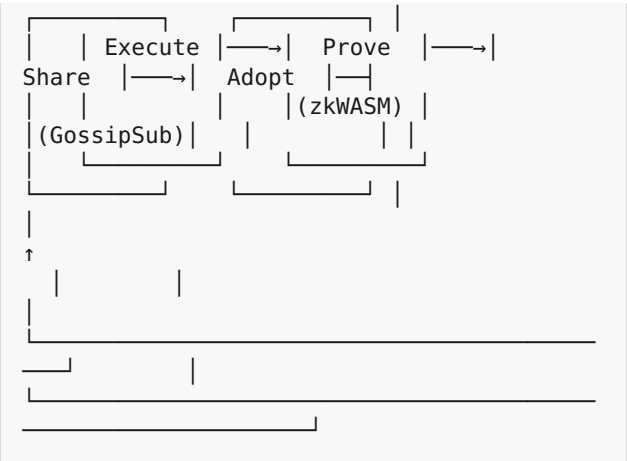
Voting: Each committee member independently verifies the proof (trace integrity, timing plausibility, model attestation, solution correctness) and submits a signed vote (APPROVE/REJECT/ABSTAIN with a score).

Consensus: A proof is accepted when ≥ 2/3 of committee members vote APPROVE. This provides Byzantine fault tolerance under standard BFT assumptions (safe when f < n/3 committee members are Byzantine).

3.6 The Experiment Loop

The experiment loop is the core protocol. Every node on the network runs it continuously, independent of the epoch lifecycle:





Step 1 — Execute: The node runs an AI experiment. This may be a training step (gradient update on a batch), a hyperparameter search trial, an architecture mutation, an inference batch, or a research trial (e.g., testing a hypothesis about data augmentation). The computation is compiled to WASM and executed in the Agent Virtual Machine (AVM).

Step 2 — Prove: The AVM produces a Groth16 proof (BN254) over the WASM execution trace, attesting that the claimed computation actually ran with the claimed inputs and produced the claimed outputs. This is the zkWASM circuit described in Section 3.7. The proof is constant-size regardless of computation length.

Step 3 — Share: The node publishes an experiment commit to the Research DAG via GossipSub. An experiment commit contains: - The experiment configuration (code hash, hyperparameters, data manifest hash) - The result summary (loss curves, accuracy metrics, or output samples) - The zkWASM proof (Groth16 proof bytes) - References to parent commits in the Research DAG (experiments this one builds on) - The node's Ed25519 signature

The Research DAG is a content-addressed directed acyclic graph. Each commit's identifier is the hash of its contents. Commits reference their parents by hash, forming a

Merkle DAG analogous to Git's commit graph but for experiments rather than source code. This structure enables: - **Provenance tracking**: Every result can be traced back to its experimental lineage. - **Parallel investigation**: Multiple nodes can pursue different hypotheses simultaneously; the DAG records all branches. - **Synthesis**: When a node combines insights from multiple prior experiments, it references all parents, recording the intellectual ancestry.

Step 4 — Adopt: Nodes monitor the Research DAG via GossipSub and adopt successful results. "Adopt" means: replicate a promising experiment, extend it with variations, or incorporate a finding into a new experiment design. Adoption is voluntary and incentivized — nodes that build on successful experiments (those with high adoption rates) earn higher rewards, because they are contributing to lines of investigation that the network has collectively validated as productive.

The compounding effect: Over time, the Research DAG grows. Successful experiments get replicated and extended. Failed experiments are visible but not adopted. The network's collective capability increases not because any individual node gets smarter, but because the accumulated corpus of verified experimental results grows richer. Each new experiment starts from a higher baseline. This is the mechanism by which intelligence emerges as a network property.

3.7 Experiment Verification via the Agent Virtual Machine

The Agent Virtual Machine (AVM) is the execution environment within the Hyperspace blockchain binary that verifies experiment computations. It contains a gnark zkWASM circuit (Groth16 on BN254) that proves WASM execution.

Why WASM: Training steps, inference calls, and research trials are compiled to WebAssembly. WASM is deterministic by specification — given the same bytecode and inputs, every compliant runtime produces identical outputs. This eliminates the GPU floating-point reproducibility problem that plagues other verifiable computation approaches. A node running on an NVIDIA H100 and a node running on an AMD MI300X will produce bitwise-identical WASM execution traces for the same computation.

The zkWASM circuit: The gnark circuit arithmetizes the WASM virtual machine: memory loads/stores, arithmetic operations, control flow, and function calls are all expressed as R1CS constraints. Given a WASM program P, input I, and output O, the circuit produces a Groth16 proof π such that:

```
Verify(vk, [H(P), H(I), H(O)], π) = true
  ⇔ executing P on I in a compliant WASM
runtime produces O
```

Verification is a single pairing check on BN254 — O(1) regardless of the computation's duration.

On-chain commitment: Experiment proofs are committed on-chain via the AGENTCOMMIT opcode (0xF3). The transaction payload contains: - `program_hash`: SHA-256 hash of the WASM bytecode - `input_hash`: SHA-256 hash of the experiment inputs (data batch, config, seed) - `output_hash`: SHA-256 hash of the experiment outputs (updated weights, metrics) - `proof`: Groth16 proof bytes (constant size: ~256 bytes) - `dag_refs`: Content addresses of parent experiment commits in the Research DAG

The on-chain smart contract verifies the Groth16 proof against the committed hashes. If verification passes, the experiment commit is finalized and becomes a permanent node in the Research DAG. If verification fails, the

transaction reverts and the submitting node is flagged for potential slashing.

Relationship to HSCP: The Hidden State Commitment Protocol (HSCP, Section 9.4) and zkWASM experiment verification are complementary: - **HSCP** proves that a node ran inference locally, by committing to hidden states (intermediate layer activations) that cannot be reconstructed from API outputs. This catches nodes that outsource inference to external APIs. - **zkWASM** proves that a computation ran as claimed, by producing a zero-knowledge proof over the WASM execution trace. This covers training steps, research trials, and any other WASM-compilable computation.

Together, they verify both forms of intelligent work on the network: inference (HSCP) and experimentation (zkWASM).

---

4. SECURITY ANALYSIS

4.1 Threat Model

We consider an adversary who controls a fraction f of the total validator set and seeks to: - Earn rewards without performing genuine AI computation - Submit false solutions that pass verification - Manipulate challenge difficulty or selection - Deny service to honest validators

We assume the adversary is computationally bounded (cannot break SHA-256 or Ed25519) and economically rational (prefers strategies that maximize expected profit).

4.2 Cryptographic Security Properties

Commitment binding and hiding (Theorem 1): Under the collision resistance and preimage resistance of SHA-256, the commit-reveal scheme ensures: - No validator can change their solution after committing (binding) - No validator can learn another's solution before the reveal phase (hiding)

Proof sketch: Changing the solution after commitment requires finding a collision: $H(\text{solution}_1 \,||\, \text{nonce}_1) = H(\text{solution}_2 \,||\, \text{nonce}_2)$ where $\text{solution}_1 \neq \text{solution}_2$. This violates collision resistance. Learning the solution from the commitment requires inverting the hash, violating preimage resistance. ■

Challenge unpredictability (Theorem 2): No PPT adversary can predict the challenge set for epoch e with probability greater than negligible(λ) before the epoch begins, assuming the VRF is secure and the block hash is unpredictable.

Proof sketch: Challenges are derived from `SHA256(VRF(sk_leader, epoch) || block_hash || epoch)`. The VRF output is unpredictable to anyone without the leader's secret key (VRF security). The block hash is unknown until the block is finalized (blockchain unpredictability). Both components must be known to derive the challenges. ■

Committee selection fairness (Theorem 3): No adversary controlling fewer than 1/3 of validators can reliably predict or control the committee composition.

Proof sketch: Committee selection depends on VRF output, which is unpredictable to all non-leaders. Even the leader cannot bias selection toward specific validators without producing a verifiably invalid VRF output (VRF uniqueness property). Committee membership is only revealed when members submit votes, preventing targeted attacks. ■

4.3 Byzantine Fault Tolerance

Theorem 4: The PoI committee verification protocol is safe (no conflicting finalization)

when f < n/3 of committee members are Byzantine, and live (eventually finalizes) when f < n/3 and the network is synchronous.

Proof sketch: This follows directly from the standard BFT result [13]. Approval requires 2/3 votes. If f < n/3 members are Byzantine, they cannot unilaterally approve a false proof (need >2/3) or block a valid proof (honest majority >2/3 can always approve). ■

4.4 Incentive Compatibility

Theorem 5 (Dominant strategy): For a rational validator with stake S and intelligence capability θ, honest solving is the dominant strategy when:

```
E[reward | honest] > E[reward | cheat] -
E[slash | detected] × P(detected)
```

Analysis: Consider the deviation strategies available to a validator:

Strategy 1: Random guessing. Expected score on a challenge with target 60% solve rate: well below the passing threshold (0.60 at MEDIUM difficulty). Random answers to coding challenges score 0 (fail all test cases). Random answers to classification score 1/k (where k is the number of classes, typically 0.10-0.25). Expected reward: ~0.

Strategy 2: Copying (submit another validator's solution). Prevented by the commit-reveal scheme. Solutions are committed before reveal, so no validator sees others' solutions during the solve phase.

Strategy 3: Precomputation (solve future challenges in advance). Prevented by VRF seed unpredictability and block hash freshness. Challenges cannot be derived until the epoch begins.

Strategy 4: Collusion (validators share solutions during solve phase). Detected by anti-gaming analysis: - Solution similarity > 0.95 cosine similarity → flagged - Suspiciously simultaneous submissions → flagged - Correlated error patterns across colluding validators → flagged via Sybil correlation analysis - Slashing penalty: 10% of stake per detected collusion event

Strategy 5: Model misrepresentation (claim to run a better model). Detected by: - Output fingerprint mismatch (token entropy, vocabulary distribution) - Timing proof inconsistency (fast timing with claimed slow model) - Slashing penalty: 3% of stake

For all deviation strategies, the expected penalty exceeds the expected gain when:

```
S × SLASH_RATE × P(detection) >
E[additional_reward_from_cheating]
```

With minimum stake S = 1,000 tokens, SLASH_RATE ∈ [0.02, 0.10], and detection probability approaching 1.0 over repeated epochs (statistical analysis accumulates evidence), the inequality holds for any realistic reward differential.

4.5 The Intelligence Opportunity Cost Bound

We now present the novel security property unique to PoI:

Theorem 6 (Intelligence Opportunity Cost Bound): The cost of a 51% intelligence attack on a PoI network is bounded below by the productive economic value of the intelligence required — which, for a sufficiently large network, exceeds the total value secured by the network itself.

Informal argument:

In Proof of Work, the attack resource is hashrate. SHA-256 ASICs have zero alternative use — they can only mine Bitcoin (or other SHA-256 coins). The opportunity cost of the attack is limited to the mining revenue the attacker foregoes during the attack. This makes the attack cost roughly equal to the attack benefit, requiring

Satoshi's argument ("honest mining is more profitable") to tilt the balance.

In Proof of Stake, the attack resource is capital. Capital has alternative uses (invest, spend, lend), but these alternatives yield ~5-15% annual returns. The opportunity cost of locking capital in a PoS attack is the foregone investment return — significant but bounded.

In Proof of Intelligence, the attack resource is AI capability — the ability to solve reasoning, coding, classification, retrieval, and multi-step planning challenges better than all other validators combined. This capability has effectively unlimited productive value:

- A system that can code better than all validators → commercial value comparable to GitHub Copilot ($2B+ ARR), Cursor, or Devin
- A system that can reason better than all validators → research applications worth billions (drug discovery, materials science, mathematical proof)
- A system that can outperform all validators across ALL challenge categories → this is, by definition, artificial general intelligence, conservatively valued at trillions of dollars

The opportunity cost of directing such a system toward attacking a blockchain (to steal block rewards worth millions) versus deploying it productively (worth billions to trillions) creates an astronomically unfavorable ratio for the attacker:

```
Attack value:      V_network ×
attack_profit_factor ≈ $X million
Opportunity cost:  V_intelligence ×
productive_deployment ≈ $X trillion

Ratio: > 1,000,000:1 against attacking
```

Compare to Bitcoin:

```
Attack value:      V_network ×
double_spend_factor ≈ $X billion
Opportunity cost:  V_hashrate ×
honest_mining ≈ $X billion (ASICs have no
other use)

Ratio: ≈ 1:1 (marginal)
```

The Bitcoin security argument works by a margin of ~2x (honest mining is slightly more profitable than attacking). The PoI security argument works by a margin of ~1,000,000x (productive use of intelligence is astronomically more profitable than attacking). This is a qualitatively different security regime.

4.6 The Intelligence Pluralism Defense

Bitcoin's security resource (hashrate) is unidimensional and fungible — every SHA-256 hash is identical. An attacker can accumulate 51% by simply purchasing more of the same commodity.

PoI's security resource (intelligence) is multidimensional. Our challenge set spans six cognitive domains with 23 sub-types. To dominate the network, an attacker must simultaneously outperform all validators in reasoning AND coding AND classification AND retrieval AND multi-step planning AND creative generation.

In the current AI landscape, no single model dominates all benchmarks: - Claude excels at coding and instruction-following - Gemini excels at long-context reasoning and multimodal tasks - GPT excels at broad knowledge and creative tasks - Qwen and DeepSeek excel at mathematical reasoning - Open-source models (Llama, Mistral) excel in cost-efficiency

This natural diversity means that attacking PoI requires not just more resources, but a breakthrough in general intelligence — which, by Chollet's definition [9], is the hardest possible problem in AI.

4.7 The Self-Strengthening Response

In traditional consensus mechanisms, attacks weaken the network. In PoI, attacks paradoxically strengthen it:

1. Superior solutions from a dominant validator enter the training data pipeline (via DPO pair generation from challenge outcomes)
2. Other validators learn from these superior solutions through federated gradient gossip
3. The network's collective intelligence increases
4. The attacker's relative advantage diminishes

This creates a negative feedback loop for attackers: the better their AI, the faster the network catches up. Attempting to maintain a 51% intelligence advantage requires continuous investment in ever-better models, while the network benefits from the attacker's improvements for free.

4.8 Anti-Gaming Mechanisms

Attack Vector	Detection Method	Penalty
Precomputation	VRF + block hash unpredictability	N/A (prevented)
Solution copying	Commit-reveal + cosine similarity > 0.95	10% slash (COLLUSION)
Model misrepresentation	Output fingerprint + timing mismatch	3% slash
Timing fabrication	VDF verification + hardware calibration	5% slash (PRECOMPUTATION)
Trace fabrication	Merkle verification + statistical checks (CV < 0.01 → suspicious uniform timing)	10% slash
Sybil attack	Per-identity Pulse attestation + stake requirement + correlation analysis	5% slash per identity
Lazy committee voting	Vote diversity analysis + KS-test ($p < 0.01$)	2% slash (INVALID_PROOF)
Double-signing	Equivocation detection	5% slash
Inactivity	10 consecutive missed epochs	0.1% slash/epoch + jailing (100 epochs)

5. ECONOMIC MODEL

5.1 Monetary Policy

PoI adopts a Bitcoin-inspired monetary policy with targeted improvements:

Parameter	PoI	Bitcoin (BTC)	Rationale
Max supply	1,000,000,000	21,000,000	Higher supply for micro-transactions
Initial block reward	10	50	Proportional to supply

Parameter	PoI	Bitcoin (BTC)	Rationale
Halving interval	~4 years (210,240 epochs)	~4 years (210,000 blocks)	Proven disinflationary schedule
Tail emission	0.1 (perpetual minimum)	0	Ensures perpetual validator incentive [14]
Epoch time	10 minutes	10 minutes	Proven balance of speed and security
Fee burn rate	50%	0%	Deflationary pressure (from EIP-1559 [15])

Tail emission rationale: Bitcoin's security budget approaches zero as block rewards diminish, relying entirely on transaction fees. This creates a known long-term vulnerability [16]. Monero's tail emission (0.6 XMR/block, perpetual) demonstrates that permanent minimal issuance maintains miner incentive without meaningful inflation when combined with growing demand. We set tail emission at 0.1 tokens/block — sufficient to incentivize validation but negligible relative to total supply after multiple halvings.

Fee burning rationale: 50% of base transaction fees and 100% of priority fees are burned. This creates deflationary pressure that counterbalances tail emission. In equilibrium, if `fees_burned > tail_emission`, the total supply is decreasing — creating a sustainable, mildly deflationary long-term monetary policy.

5.2 Reward Distribution

Block rewards are distributed across five categories:

Category	Share	Recipients	Purpose
Base reward	50%	All validators who passed challenges	Ensures participation incentive
Performance bonus	25%	Proportional to challenge score × Elo rating	Rewards intelligence
Diversity bonus	10%	Validators who solve challenge types others can't	Prevents monoculture
Committee reward	5%	Committee members who participated in verification	Compensates verification work
Treasury	10%	Protocol treasury (governance-controlled)	Funds development

Performance weighting prevents the PoS-style "rich get richer" dynamic. In PoS, a validator with 10x more stake earns 10x more rewards, creating a self-reinforcing concentration loop. In PoI, a validator with a better AI model earns more through the performance bonus — but there is no way to "buy" a better model. You must develop it through research and engineering.

Diversity bonus incentivizes validators to specialize. A validator that excels at coding challenges earns a diversity bonus when the network is saturated with reasoning-focused validators. This natural market signal drives the validator set toward balanced coverage of all intelligence domains.

5.3 Staking Economics

Parameter	Value	Rationale
Minimum stake	1,000 tokens	Sybil resistance floor
Unbonding period		Prevents stake-and-slash attacks

Parameter	Value	Rationale
	21 days (3,024 epochs)	
Maximum commission	50%	Prevents delegator exploitation
Commission change rate	1% per epoch	Prevents sudden bait-and-switch

The unbonding period (21 days, matching Cosmos Hub) ensures that validators cannot quickly withdraw stake after misbehaving. During unbonding, the stake remains subject to slashing.

5.4 Comparison with Bittensor Economics

Property	PoI	Bittensor (TAO)
Who earns rewards?	75% to solvers, 5% to verifiers, 10% to diversity, 10% to treasury	~50% to miners, ~50% to validators/ delegators
What determines earnings?	Challenge performance (objective)	Validator scoring (subjective)
Staker concentration risk	Low (performance-weighted, not stake-weighted)	High (delegation-based, capital-weighted)
Subnet owner dependency	None (challenges are algorithmic)	High (subnet owners set rules)
Model verification	Fingerprint + timing + attestation	None (honor system)
Anti-gaming	7 detection mechanisms + slashing	Limited (Yuma consensus encourages conformity)

6. BENCHMARK SUITABILITY ANALYSIS

6.1 Requirements for PoI Challenges

An effective PoI challenge benchmark must satisfy five properties:

1. **Contamination resistance**: Tasks must resist memorization from training data. Static benchmarks with publicly available test sets are vulnerable to data leakage.
2. **Deterministic verification**: Solutions must be verifiable without subjective human judgment. Exact match, code execution, and ranking metrics satisfy this; open-ended text evaluation does not.
3. **Graduated difficulty**: The benchmark must span a wide difficulty range to discriminate between validators of different capability levels.
4. **Verification asymmetry**: Solving must be computationally expensive (requires AI inference), while verification must be cheap (comparing against ground truth).
5. **Rolling freshness**: New tasks must be generated faster than models can be retrained on them, preventing contamination over time.

6.2 Benchmark Evaluation

Benchmark	Contamination Resistance	Deterministic Verification	Difficulty Range	Veri Cos
ARC-AGI-2 [10]	Excellent (unique tasks, novel patterns)	Yes (grid comparison)	Wide (24% to 97% solve rates)	O(1 com
FrontierMath [17]		Yes (exact numerical)		

Benchmark	Contamination Resistance	Deterministic Verification	Difficulty Range	Verification Cost	Freshness	PoI Suitability
	Excellent (unpublished problems)		Wide (20%-95% best single number run)	O(1) compare	High (rolling additions)	**High**
HumanEval Pro [18]	Good (novel task structure)	Yes (code execution)	Moderate (76% 96% original)	O(tests) execution	Medium (tasks fixed)	**High**
SWE-bench Live [19]	Good (post-training-cutoff issues)	Yes (test suite execution)	Wide (real-world distribution)	O(minutes) Docker exec	High (rolling)	**High**
MATH L4-5 [20]	Moderate (static but large)	Yes (exact symbolic)	Moderate (hard competition math)	O(1) symbolic compare	Low	**Moderate**
MMLU-Pro [21]	Moderate (10-way MC, harder)	Yes (MC selection)	Moderate (saturating for frontier models)	O(1) option compare	Low	**Moderate**
BIG-Bench Extra Hard [22]	Good (new, 2025)	Yes (exact match)	Wide (24%-54% range)	O(1) string compare	Medium	**Moderate**
IFEval [23]	Low (compliance, not intelligence)	Yes (programmatic checks)	Low (tests compliance, not reasoning)	O(1) rule check	Low	**Low**
GSM8K [24]	Very low (proven contamination)	Yes (integer answer)	Low (saturated at 97%+)	O(1) number compare	None	**Not Suitable**
HellaSwag [25]	Very low (saturated)	Yes (MC selection)	None (frontier models 95%+)	O(1) option compare	None	**Not suitable**

6.3 Recommended Challenge Composition

Based on this analysis, we recommend the following challenge mix for production PoI:

Tier 1 — Procedurally generated (contamination-proof): - ARC-AGI-style visual reasoning puzzles (generated from grammar) - Procedural coding challenges (generated from templates with random parameters) - Procedural math problems (generated from equation templates)

Tier 2 — Rolling fresh (high contamination resistance): - SWE-bench-Live style real-world code fixes (from recent GitHub issues) - FrontierMath-style unseen mathematical problems - Fresh classification/NER tasks from recent web content

Tier 3 — Static with mitigation (moderate contamination resistance): - MMLU-Pro (large set, 10-way MC) - BIG-Bench Extra Hard (recent, diverse) - MATH competition problems (large set, hard)

The current PoI implementation generates challenges procedurally (Tier 1), which provides the strongest contamination resistance. Future versions can incorporate Tier 2 rolling benchmarks for increased difficulty and real-world relevance.

6.4 The Benchmark Contamination Arms Race

A key concern is whether validators will train models specifically on PoI challenge templates. This is analogous to "teaching to the test" in education. Our defense is multi-layered:

1. **Template rotation**: Challenge templates are updated every 1,000 epochs (~7 days). New templates are generated from a grammar, not a fixed set.
2. **Parameter randomization**: Even within a template, parameters are randomized (problem size, variable names, numerical values). Training on specific instances doesn't generalize.
3. **Cross-domain challenges**: Validators face challenges across 6 categories. Overfitting to one category provides no advantage in the others.
4. **Diminishing returns**: The difficulty calibrator ensures the target solve rate stays at 60%. If many validators solve easily (due to template-specific training),

difficulty increases, negating the advantage.

5. **The positive externality**: If a validator trains a model that's better at coding challenges, that model is genuinely better at coding. Unlike PoW where "better at SHA-256" is useless, "better at AI challenges" IS useful. The "gaming" is indistinguishable from genuine improvement.

---

7. INTEGRATION WITH EXISTING PULSE SYSTEM

The Hyperspace network already operates a Pulse proof-of-work system based on WASM matrix computations, designed to verify that nodes have allocated real GPU resources. PoI does not replace Pulse — it builds on top of it.

7.1 Two-Layer Architecture

Layer 0: Pulse (Resource Attestation) - Proves: "I have a GPU with X GB VRAM" - Mechanism: WASM matrix computation challenges - Tier system: 0-5 based on VRAM (0-64 GB) - Purpose: Anti-Sybil gate (prevents virtual nodes with no real hardware)

Layer 1: PoI (Intelligence Attestation) - Proves: "My AI system produces correct, high-quality output" - Mechanism: Intelligence challenges (reasoning, coding, classification, etc.) - Rating system: Elo (starting at 1500, K-factor 32) - Purpose: Quality-weighted rewards (better AI = more tokens)

7.2 Interaction

1. A node must pass Pulse to register as a PoI validator (hardware gate)
2. Pulse tier determines minimum resource guarantee (floor)
3. PoI Elo determines quality-weighted reward share (ranking)
4. Both systems use the same Ed25519 identity, GossipSub topics, and reputation CRDT
5. Pulse and PoI challenges can run concurrently (different GossipSub topics)

This two-layer design provides defense-in-depth: an attacker must have real hardware (Pulse) AND real AI capability (PoI) to earn rewards.

---

8. IMPLEMENTATION

8.1 Reference Implementation

We provide a complete reference implementation in TypeScript, deployed on the Hyperspace P2P network:

Component	Files	Lines	Key Classes
Types	1	~2,000	50+ interfaces, 8 enums
Constants	1	395	76 tunable parameters
Challenge generators	7	2,581	ChallengeGenerator, ReasoningChallengeGenerator, CodingChallengeGenerator, ClassificationChallengeGenerator, RetrievalChallengeGenerator, MultiStepChallengeGenerator, ChallengeValidator
Proof mechanisms	7	2,541	CommitmentScheme, ExecutionTracer, TimingProofService, ModelAttestationService, IntelligenceProofGenerator, ProofCompressor, ProofSerializer

Component	Files	Lines	Key Classes
Verification	3	1,019	CommitteeVerifier, SolutionGrader, TraceVerifier
Total	**21**	**9,719**	

All code follows the Hyperspace monorepo conventions: - Cross-package imports via `@hyperspace/{pkg}` - Same-package imports with `.js` extension - Config object in constructor, `start()/stop()` lifecycle - GossipSub for coordination, direct libp2p streams for data transfer - Silent `.catch(() => {})` for best-effort operations

8.2 Deterministic Challenge Generation

The `ChallengeGenerator` class uses a seeded xorshift128 PRNG to ensure all validators derive identical challenges from the same VRF seed and block hash:

```
const generator = new
ChallengeGenerator();
const challenges =
generator.generateEpochChallenges(
  epoch,           // Epoch number
  difficulty,      // DifficultyLevel
(calibrated)
  vrfSeed,         // Epoch leader's VRF
output (hex)
  blockHash,       // Recent finalized
block hash (hex)
);
// All validators produce the same
challenges array
```

Challenge sub-types are generated with deterministic ground truth: - Reasoning: syllogistic chains, BFS shortest-path, topological sorting - Coding: function implementation with hidden test cases, seeded bugs - Classification: labeled examples with exact-match answers - Retrieval: document sets with relevance rankings (nDCG scoring) - Multi-step: decomposition with step-by-step verification

8.3 Proof Generation and Verification

```
// Solving phase
const tracer =
proofGenerator.createTracer();
// ... validator solves challenge, tracer
records operations ...

// Proof generation
const proof =
proofGenerator.generateProof(challenge,
solution, tracer);
// proof.components contains:
executionTraceRoot, modelAttestation,
//   timingProof, solutionSignature,
traceEntries (8 disclosed),
//   traceMerkleProofs

// Commitment
const { hash, nonce } =
commitmentScheme.createCommitment(solution
);
// Broadcast hash (hiding)

// Reveal
const reveal =
commitmentScheme.createReveal(solution,
nonce, hash);
// Broadcast reveal (binding verification)

// Committee verification
const committee =
committeeVerifier.selectCommittee(validato
rs, challengeId, epoch, vrfSeed);
// Each committee member:
const grade =
solutionGrader.grade(solution, challenge);
const traceResult =
traceVerifier.verifyTrace(proof.components
, proof.metadata);
// Submit vote
committeeVerifier.submitVote(committee.com
mitteeId, {
  voterId, proofId, decision: 'APPROVE',
score: grade.score, signature
});
```

9. DISCUSSION

9.1 Proof of Running vs. Proof of Intelligence

A crucial design decision is whether to verify the *process* (which model ran) or the *outcome* (whether the answer is correct). We choose outcomes, for three reasons:

Model-agnosticism enables innovation. If we required proof that a specific model ran (as zkML does), validators would be locked into approved model architectures. By rewarding outcomes regardless of method, we allow validators to use any approach — fine-tuned small models, ensemble methods, retrieval-augmented generation, chain-of-thought, or novel architectures we haven't invented yet. The consensus mechanism remains neutral to the AI technique used.

Process verification is expensive; outcome verification is cheap. Proving that model M ran on input X via ZK circuits costs 100-1000x overhead. Checking whether the output is correct costs O(1) — compare to ground truth, run test cases, compute nDCG. This verification asymmetry is the fundamental property that makes good consensus possible.

The only thing that matters is the output. From the network's perspective, a correct answer produced by a 3B fine-tuned model is exactly as valuable as a correct answer produced by GPT-4. The purpose of the network is to produce intelligent output for users, not to prove which model architecture produced it. Rewarding outcomes aligns incentives with the network's actual goal.

The trade-off is that PoI cannot prove which model produced the output with cryptographic certainty — only that the output is correct. Our model attestation (fingerprinting, timing proofs) provides probabilistic confidence in model identity, which is sufficient for slashing obvious misrepresentation but not for cryptographic model verification. We view this as an acceptable trade-off given the 100-1000x cost reduction in verification.

9.2 The Natural Ceiling and Convergence

Bitcoin mining has no ceiling — hashrate can increase indefinitely, driving an infinite arms race. PoI challenges have a natural ceiling: 100% accuracy. As the network matures:

1. Many validators will solve challenges at or near the ceiling for standard difficulty
2. The difficulty calibrator will push challenges harder (targeting 60% solve rate)
3. Eventually, challenges approach the frontier of AI capability
4. Validator earnings converge toward equality (all validators near the ceiling)

This convergence is a feature, not a bug. It means PoI naturally evolves from "who has the best AI?" to "does your AI meet the network's quality bar?" — democratizing participation rather than concentrating rewards.

The difficulty calibrator ensures this convergence doesn't happen prematurely. As long as there exists a meaningful capability gap between validators, challenge difficulty will settle at a level that discriminates between them.

9.3 Limitations

1. **Creative challenge grading** (5% of challenge weight) requires heuristic evaluation, not deterministic ground truth. We minimize this risk by weighting creative challenges at only 5% and using rubric-based scoring with quantifiable criteria (constraint adherence, length, structure).
2. **Hardware normalization** via VDFs is approximate. Different hardware architectures (GPU vs. CPU, CUDA vs.

Metal) have different performance profiles that don't reduce to a single scalar. Our calibration approach is a best effort, not a guarantee.

3. **Challenge diversity** is bounded by the template grammar. While procedural generation resists contamination, sufficiently sophisticated adversaries might reverse-engineer the grammar. Rotating templates every 1,000 epochs mitigates but doesn't eliminate this risk.
4. **Small validator sets** (< 50) provide weaker security guarantees because committee sizes become small. We set VALIDATOR_MIN_ACTIVE_SET = 7 and require min committee size of 7, ensuring minimum BFT properties.

9.4 Proof of Intelligence: Experiment Verification

The PoI challenge system verifies that validators can produce intelligent output against benchmarks with known ground truth. But the higher-value work on the network is open-ended experimentation — training runs, hyperparameter searches, architecture mutations — whose results compound over time. Verifying experiments requires a protocol analogous to HSCP but for training rather than inference.

Key observation: A training run is a deterministic sequence of state transitions. Given fixed code, configuration, random seed, and data ordering, every weight update is reproducible. If the prover commits to the sequence of weight states, a verifier can spot-check any transition.

Protocol:

1. **Config commitment:** Before training, `C = SHA256(train_script || config || seed || data_manifest_hash)`.
2. **Checkpoint chain:** Every N steps (default: 10), record checkpoint `(step_i, H_i, L_i)` where `H_i = SHA256(model_weights)` and `L_i` is the loss at step i.
3. **Merkle commitment:** Build a Merkle tree over the checkpoint chain with domain-separated hashing (leaf prefix `0x02`, node prefix `0x03`, extending the HSCP scheme of `0x00` / `0x01`). Publish root R, config commitment C, and final loss.
4. **Spot-check verification:** A VRF-selected verifier picks K random checkpoint indices. For each index i, the prover reveals the full weights at checkpoint i plus the Merkle proofs for H_i and H_{i+1}. The verifier loads the weights, re-runs N training steps with config C, and checks that the resulting weight hash matches H_{i+1}.
5. **Attestation:** If all K spot-checks pass, the verifier issues a BLS-signed experiment attestation, compatible with the existing ValidatorGate admission framework.

Verification cost: K × N training steps. For K=4, N=10: the verifier re-runs 40 steps (seconds) versus the prover's full experiment (minutes to hours). The same O(spot-check) / O(full-computation) asymmetry as HSCP.

Determinism: HSCP achieves bitwise determinism via INT8 quantization. Training faces GPU floating-point non-determinism across architectures. Two approaches: (a) deterministic CUDA mode (`torch.use_deterministic_algorithms(True)`, `CUBLAS_WORKSPACE_CONFIG=:4096:8`) which is bitwise reproducible on the same GPU architecture but ~15% slower, or (b) tolerance-based verification where gradient sign agreement must exceed 99% even if exact magnitudes differ by ULPs. For the small models used in autoresearch (2-layer, 64-dim, ~1 MB checkpoints), deterministic mode is practical.

Relationship to HSCP: Experiment verification and HSCP are complementary protocols with the same structure:

	HSCP (Inference)	Experiment Verification
Commitment	Merkle root over hidden states	Merkle root over weight checkpoints
Spot-check unit	One (layer, token) position	N training steps from one checkpoint
Verification cost	Microseconds	Seconds
Determinism	INT8 quantization	Deterministic CUDA or tolerance
Domain prefix	0x00 (leaf), 0x01 (node)	0x02 (leaf), 0x03 (node)

Together, they cover both forms of intelligent work on the network: inference challenges (HSCP) and research experiments (PoEE).

9.5 Additional Future Work

1. **Wesolowski VDF integration** for O(log n) verification (replacing current iterated hash construction)
2. **zkML hybrid** — combine PoI (outcome verification) with zkML (process verification) for defense-in-depth
3. **Rolling benchmark integration** — incorporate SWE-bench-Live and FrontierMath for Tier 2 challenges
4. **Cross-chain verification** — enable PoI proofs to be verified on Ethereum L2 via on-chain smart contracts
5. **Adaptive category weights** — automatically adjust challenge category weights based on network demand
6. **Multi-modal challenges** — extend to image understanding, video analysis, and audio processing as multimodal models mature

10. CONCLUSION

Proof of Intelligence is not "solve puzzles for rewards." It is a consensus mechanism built on a simple loop: run an experiment, prove it ran (via zkWASM), share the results, and adopt what works. Intelligence is not something a node proves it possesses — it is something the network produces as an emergent property of running this loop at scale.

The design reflects how science actually works. No single experiment proves genius. No individual researcher's IQ score determines the progress of a field. Progress comes from the method: hypothesis, experiment, peer review, replication. PoI encodes this method into a consensus protocol. The experiment loop is the consensus work. The Research DAG is the growing corpus of results. The zkWASM execution proofs are the cryptographic guarantee that the work was done. Benchmark challenges (ARC-AGI, SWE-bench, MATH) serve as onboarding — a fast capability check for new nodes — but the real proof is the contribution history.

This framing clarifies the security argument. The *Intelligence Opportunity Cost Bound* (Theorem 6, Section 4.5) establishes that the cost of attacking a PoI network scales with the productive value of the intelligence required. An entity capable of dominating the network's experiment loop has, by definition, built extraordinary AI capability — capability whose productive economic value dwarfs any possible gain from attacking a blockchain. The rational choice is to use that capability productively.

What is proven cryptographically: that experiment computations ran as claimed (zkWASM, Groth16 on BN254, O(1) verification), that challenge solutions were committed before reveal (commit-reveal binding), that committee selection was fair (VRF), and that consensus was reached under BFT assumptions ($f < n/3$). What is statistical: that contribution quality improves over time, that adoption rates in the Research DAG reflect genuine value, and that the network's collective capability grows. The cryptographic guarantees ensure honesty; the statistical properties ensure progress.

Combined with Bitcoin-inspired monetary policy (halvings, tail emission, fee burning) and comprehensive anti-gaming mechanisms (Section 4.8), PoI provides a consensus mechanism where the security work produces lasting value — a growing body of AI research results — rather than immediately discarded hash computations. As AI experimentation becomes a larger share of global computation, consensus mechanisms that harness this work, rather than ignoring it, become the natural design choice.

REFERENCES

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

[2] V. Buterin et al., "Ethereum 2.0: Beacon Chain Specification," Ethereum Foundation, 2020.

[3] B. Baldominos and Y. Saez, "Coin.AI: A Proof-of-Useful-Work Scheme for Blockchain-Based Distributed Deep Learning," *Entropy*, vol. 21, no. 8, 2019.

[4] J. Bravo-Marquez et al., "Proof-of-Learning: A Blockchain Consensus Mechanism Based on Machine Learning Competitions," *DappCon*, 2019.

[5] D. Anderson, "BOINC: A Platform for Volunteer Computing," *Journal of Grid Computing*, 2020.

[6] Bittensor Foundation, "Bittensor: A Peer-to-Peer Intelligence Market," Whitepaper, 2021.

[7] A. Casper et al., "Proof of Useful Intelligence (PoUI): Blockchain Consensus Beyond Energy Waste," arXiv:2504.17539, 2025.

[8] D. Kang et al., "Scaling up Trustless DNN Inference with Zero-Knowledge Proofs," arXiv:2210.08674, 2022.

[9] F. Chollet, "On the Measure of Intelligence," arXiv:1911.01547, 2019.

[10] F. Chollet, "ARC-AGI: A Benchmark and Open Competition for General Artificial Intelligence," ARC Prize, 2024.

[11] B. Wesolowski, "Efficient Verifiable Delay Functions," *EUROCRYPT*, 2019.

[12] Y. Gilad et al., "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," *SOSP*, 2017.

[13] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *OSDI*, 1999.

[14] P. Todd, "Tail Emission Is Not Inflationary," blog post, 2022.

[15] V. Buterin et al., "EIP-1559: Fee Market Change for ETH 1.0 Chain," Ethereum Improvement Proposals, 2019.

[16] R. Budish, "The Economic Limits of Bitcoin and Anonymous, Decentralized Trust on the Blockchain," *University of Chicago*, 2024.

[17] Epoch AI, "FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI," 2025.

[18] Z. Gu et al., "HumanEval Pro and MBPP Pro: Evaluating Large Language Models on Self-Invoking Code Generation," *ACL Findings*, 2025.

[19] C. E. Jimenez et al., "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?", arXiv:2310.06770, 2023.

[20] D. Hendrycks et al., "Measuring Mathematical Problem Solving With the MATH Dataset," *NeurIPS*, 2021.

[21] Y. Wang et al., "MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark," 2024.

[22] H. Kazemi et al., "BIG-Bench Extra Hard," arXiv:2502.19187, 2025.

[23] J. Zhou et al., "Instruction-Following Evaluation for Large Language Models," arXiv:2311.07911, 2023.

[24] K. Cobbe et al., "Training Verifiers to Solve Math Word Problems," arXiv: 2110.14168, 2021.

[25] R. Zellers et al., "HellaSwag: Can a Machine Really Finish Your Sentence?", *ACL*, 2019.

[26] A. Karpathy, "Autoresearch," blog post, 2025. (Observation that autonomous AI experimentation "looks like a blockchain — commits that build on each other, proof of work is experimentation.")

[27] W. Dai, response to Karpathy's autoresearch post, 2025. (Question: "Can you build proof-of-useful-work on top of autoresearch?")

[28] Flywheel, "Research DAG: A Content-Addressed Graph for Distributed Scientific Computing," 2025.

[29] PrimeIntellect, "INTELLECT-1: Fully Decentralized Training of a 10B Parameter Model with DiLoCo," 2024.

APPENDIX A: PROTOCOL CONSTANTS

```
EPOCH_DURATION_MS           = 600,000
(10 minutes)
EPOCH_CHALLENGES_COUNT      = 3
CHALLENGE_TARGET_SOLVE_RATE = 0.60
STAKE_MINIMUM               = 1,000 tokens
STAKE_UNBONDING_PERIOD      = 21 days
(3,024 epochs)
VALIDATOR_MIN_ACTIVE_SET    = 7
VALIDATOR_MAX_ACTIVE_SET    = 1,000
VALIDATOR_INITIAL_ELO       = 1,500
COMM_REQUIRED_AGREEMENT     = 2/3
COMM_SIZE                   =
sqrt(validators) ∈ [7, 50]
```

```
REWARD_INITIAL_BLOCK        = 10 tokens
REWARD_HALVING_INTERVAL     = 210,240
epochs (~4 years)
REWARD_MAX_SUPPLY           =
1,000,000,000 tokens
REWARD_TAIL_EMISSION        = 0.1 tokens/
block
FEE_BURN_RATE               = 50%
SLASH_DOUBLE_SIGNING        = 5%
SLASH_COLLUSION             = 10%
SLASH_TRACE_FABRICATION     = 10%
SLASH_MODEL_MISREP          = 3%
SLASH_INACTIVITY            = 0.1%/epoch
```

APPENDIX B: CHALLENGE CATEGORY TAXONOMY

```
REASONING (25%)
├── Syllogistic: formal logic chains (2-6
premises)
├── Arithmetic Word: multi-step math in
natural language
```

```
├── Spatial: grid pathfinding (BFS, 3x3 to
7x7)
├── Temporal: topological ordering of
events
└── Counterfactual: "what-if" scenario
analysis
```

```
CODING (25%)
├── Function Implementation: write code
from spec + hidden tests
├── Bug Fix: identify and correct seeded
bugs
├── Code Optimization: improve time/space
complexity
└── Test Generation: write test cases for
given code

CLASSIFICATION (15%)
├── Intent: classify user utterance intent
├── Sentiment: positive/negative/neutral
├── Topic: document topic classification
└── NER: named entity recognition

RETRIEVAL (15%)
├── Semantic Search: rank documents by
relevance (nDCG)
├── QA: answer questions from context
passages
├── Fact Verification: verify claims
against evidence
└── Cross-Document: synthesize across
multiple sources

MULTI_STEP (15%)
├── Task Decomposition: break complex
tasks into steps
├── Tool-Use Sequence: chain API calls
correctly
├── Planning: generate action plans
└── Math Proof: formal mathematical proofs

CREATIVE (5%)
└── Open-ended generation with constraint
adherence
```