Proof of AI: An Open Protocol for Decentralized AI Mining

Bitcoin-Inspired Token Economics with Quantum-Safe Security

Lux Network Foundation[*]
Hanzo AI[†]
Zoo Labs Foundation[‡]

Version 1.0 — December 2025

Abstract

This paper presents **Proof of AI (PoAI)**, an open protocol enabling anyone with GPU compute to mine AI tokens across the Lux ecosystem. Inspired by Bitcoin's elegant simplicity, PoAI establishes a fixed supply of 1 billion AI tokens per chain, with mining rewards halving on a schedule aligned with Bitcoin's halvings. The protocol leverages NVIDIA's NVTrust for hardware attestation, ML-DSA quantum-safe signatures for long-term security, and Quasar consensus for instant finality. Unlike traditional mining, PoAI rewards *useful* AI compute—inference, training, and research—while preventing double-spend through cryptographic chain-binding. The Teleport bridge enables seamless cross-chain liquidity across Hanzo EVM, Zoo EVM, and Lux C-Chain, with a novel global supply cap that only increases as new chains join the network.

Contents

[*]Technical Contact: team@lux.network
[†]AI Infrastructure: team@hanzo.ai
[‡]DeSci Research: team@zoo.ngo

1 Introduction

1.1 The Vision: Open AI for Everyone

The artificial intelligence revolution has been captured by centralized corporations. Access to frontier models requires expensive API subscriptions, training compute is controlled by hyperscalers, and the economic benefits of AI accrue to shareholders rather than contributors.

Proof of AI (PoAI) democratizes this landscape by creating an open protocol where:

1. **Anyone with a GPU can participate** — from a single RTX 4090 to data center H100 clusters
2. **Miners earn AI tokens** for providing useful compute (inference, training, research)
3. **Users pay for services** with AI tokens, creating a self-sustaining economy
4. **Supply is fixed and predictable** — 1 billion tokens per chain, halving like Bitcoin
5. **Cross-chain liquidity** enables tokens to flow freely via Teleport

1.2 Design Philosophy

PoAI draws inspiration from Bitcoin's elegant simplicity while adapting to AI compute requirements:

Aspect	Bitcoin	Proof of AI
Work	SHA-256 hashing	AI inference/training
Supply	21M BTC	1B AI per chain
Halving	Every 210,000 blocks	Every 210,000 blocks
Finality	60 minutes (6 confirms)	500ms (Quasar)
Signatures	ECDSA	ML-DSA (quantum-safe)
Hardware	ASICs	GPUs with NVTrust

1.3 Core Innovation: Chain-Bound AI Work

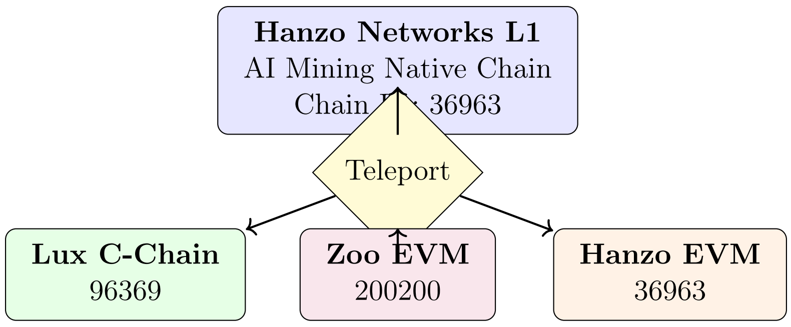
The fundamental innovation of PoAI is **chain-binding**: each unit of AI work is cryptographically committed to a specific chain *before* the compute runs. This prevents "copy-paste mining" where the same work is submitted to multiple chains.

Definition 1.1 (Chain-Bound Work). *A unit of AI work W is chain-bound if and only if:*

1. *The target chain ID c is included in the work context before computation*
2. *The GPU's NVTrust enclave signs a receipt including c*
3. *Each chain maintains a spent set preventing double-minting*

1.4 Ecosystem Overview

PoAI operates across three primary chains in the Lux ecosystem:



1.5 Document Structure

This paper is organized as follows:

- **Section 2**: Token economics, supply schedule, halving mechanism
- **Section 3**: Proof of AI consensus and reward calculation
- **Section 4**: NVTrust chain-binding and double-spend prevention
- **Section 5**: Difficulty adjustment algorithm
- **Section 6**: Market dynamics and payment for services
- **Section 7**: Multi-chain mining with Teleport
- **Section 8**: EVM contracts for reward claiming
- **Section 9**: Security analysis and quantum safety
- **Section 10**: Conclusion and future work
- **Section 11**: Shielded mining via zero-knowledge proofs

2 Token Economics

2.1 Design Philosophy: Bitcoin for AI

AI Token applies Bitcoin's proven economic model to AI compute:

Property	**Bitcoin**	**AI Token**
Total Supply	21M BTC	1B AI per chain
Initial Reward	50 BTC/block	50 AI/block
Halving Interval	210,000 blocks	210,000 blocks
Halving Period	∼4 years	∼4 years
Time to 50% mined	∼4 years	∼4 years
Time to 99% mined	∼27 years	∼27 years
Useful Work	Hash (wasteful)	AI Compute (useful)

2.2 Global Supply Model

$$S_{\text{chain}} = 1{,}000{,}000{,}000 \text{ AI (1B per chain)} \tag{1}$$

Phase	Chains	Total Supply	Market Access
Launch	10	10B AI	$100M depth
Growth	100	100B AI	$1B depth
Scale	1,000	1T AI	$10B depth

Key Insight: Each new chain adds 1B AI supply, expanding the compute market. Communities vote on which chains to add next.

2.3 Launch Chains (10)

Chain	ID	Supply	LP Depth	DEX	Bridge
Lux C-Chain	96369	1B	$10M	LuxSwap	Warp
Hanzo EVM	36963	1B	$10M	HanzoSwap	Warp
Zoo EVM	200200	1B	$10M	ZooSwap	Warp
Ethereum	1	1B	$10M	Uniswap V3	Teleport
Base	8453	1B	$10M	Uniswap V3	Teleport
BNB Chain	56	1B	$10M	PancakeSwap	Teleport
Avalanche	43114	1B	$10M	TraderJoe	Teleport
Arbitrum	42161	1B	$10M	Uniswap V3	Teleport
Optimism	10	1B	$10M	Uniswap V3	Teleport
Polygon	137	1B	$10M	Uniswap V3	Teleport
Total	—	**10B**	**$100M**	—	—

2.4 Precise Launch Pricing

2.4.1 Current GPU Compute Market (Dec 2025)

Provider	H100 $/hr	Type
AWS p5.48xlarge	$3.90	Hyperscaler
Azure NC H100 v5	$6.98	Hyperscaler
Lambda Labs	$2.99	Specialized
RunPod Secure	$2.39	Decentralized
Hyperbolic	$1.49	Decentralized
Reserved (1yr)	$1.85	Committed
Market Average	**$2.50**	—

2.4.2 Token Pricing Formula

Define: **1 AI = 1 GPU-hour of compute value**

$$P_{\text{launch}} = P_{\text{market}} \times \text{bootstrap\_discount} \tag{2}$$

Discount	AI Price	vs Market	Strategy
0%	$2.50	100%	Equilibrium
50%	$1.25	50%	Conservative
90%	$0.25	10%	Aggressive
96%	$0.10	4%	Bootstrap
99%	$0.025	1%	Extreme bootstrap

Recommended Launch Price: $\mathbf{\$0.10}$/AI (96% discount)
Rationale:

- Creates massive demand (25x cheaper than market)
- Allows 25x price appreciation to equilibrium
- \$10M LP depth = meaningful liquidity
- Attractive for early miners and users

2.4.3 LP Pool Mathematics

For Uniswap V2 constant product AMM:

$$k = x \times y \quad \text{where } x = \text{AI}, \quad y = \text{ETH} \tag{3}$$

$$P = \frac{y}{x} \times P_{\text{ETH}} \tag{4}$$

For \$10M total liquidity at \$0.10/AI:

$$\text{AI value} = \text{ETH value} = \$5\text{M} \tag{5}$$
$$\text{AI amount} = \$5\text{M}/\$0.10 = 50\text{M AI} \tag{6}$$
$$\text{ETH amount} = \$5\text{M}/\$4,000 = 1,250 \text{ ETH} \tag{7}$$

Chain	AI in LP	Pair Token	Pair Amount
Ethereum	50M	ETH	1,250 ETH
Base	50M	ETH	1,250 ETH
Arbitrum	50M	ETH	1,250 ETH
Optimism	50M	ETH	1,250 ETH
BNB Chain	50M	BNB	7,350 BNB
Avalanche	50M	AVAX	119,000 AVAX
Polygon	50M	MATIC	8.3M MATIC
Lux	50M	LUX	TBD
Hanzo	50M	LUX	TBD
Zoo	50M	LUX	TBD

2.5 Bitcoin-Aligned Halving Schedule

2.5.1 Supply Distribution Formula

Total supply per chain following Bitcoin's geometric series:

$$S = \sum_{i=0}^{\infty} 210000 \times R_0 \times 2^{-i} = 420000 \times R_0 \tag{8}$$

For $S = 1,000,000,000$ AI:

$$R_0 = \frac{1,000,000,000}{420,000} = 2,381 \text{ AI per block} \tag{9}$$

2.5.2 Block Time Analysis

Block Time	Halving Period	50% Mined	99% Mined	Annual Blocks
10 min (Bitcoin)	4 years	4 years	27 years	52,560
2 sec (Quasar)	4.86 days	4.86 days	32 days	15,768,000
12 sec (Ethereum)	29 days	29 days	195 days	2,628,000
1 min (Balanced)	**146 days**	**146 days**	**2.7 years**	**525,600**

Recommendation: Use 1-minute effective block time for mining rewards:

- Halving every ∼146 days (vs 4.86 days with 2-sec blocks)
- 50% mined in first ∼5 months
- 99% mined in ∼2.7 years
- Provides meaningful early miner incentives
- Transitions to fee-based model within reasonable timeframe

2.5.3 Adjusted Initial Reward

With 1-minute blocks (525,600 blocks/year):

$$R_0 = \frac{1,000,000,000}{420,000} = 2,381 \text{ AI per block} \tag{10}$$

But with 2-second blocks (to match Quasar consensus), scale the interval:

$$\text{Halving Interval} = 210,000 \times 30 = 6,300,000 \text{ blocks (2-sec)} \tag{11}$$

$$R_0 = \frac{1,000,000,000}{2 \times 6,300,000} \approx 79.4 \text{ AI per block} \tag{12}$$

2.6 Halving Timeline (2-sec blocks, 4-year halving)

Era	Period	Reward	Era Supply	Cumulative
1	Years 0–4	79.4 AI	500M	50%
2	Years 4–8	39.7 AI	250M	75%
3	Years 8–12	19.85 AI	125M	87.5%
4	Years 12–16	9.92 AI	62.5M	93.75%
5	Years 16–20	4.96 AI	31.25M	96.875%
⋮	⋮	⋮	⋮	⋮
10	Years 36–40	0.155 AI	0.98M	99.9%

2.7 Game Theory & Price Discovery

2.7.1 Bootstrap Phase: 96% Discount

Starting at \$0.10/AI (vs \$2.50 market):

1. **Demand Creation**: Users get 25x more compute per dollar
2. **Miner Incentive**: Mine AI, sell at discount, still profitable
3. **Price Pressure**: As demand exceeds supply, price rises
4. **Equilibrium**: Price approaches market rate as compute fills

2.7.2 Price Appreciation Mechanics

$$P(t) = P_0 \times \left(1 + \frac{D(t) - S(t)}{L}\right) \tag{13}$$

where:

- P_0 = initial price (\$0.10)
- $D(t)$ = cumulative demand
- $S(t)$ = cumulative supply (mining)
- L = LP liquidity depth

2.7.3 Expected Price Trajectory

Phase	Time	Price	vs Market	Driver
Launch	Day 0	\$0.10	4%	LP seed
Bootstrap	Month 1	\$0.25	10%	Early demand
Growth	Month 6	\$0.50	20%	Mining + usage
Adoption	Year 1	\$1.00	40%	Network effects
Maturity	Year 2+	\$2.00+	80%+	Equilibrium

2.8 Miner Economics

2.8.1 GPU Mining Profitability

At launch (\$0.10/AI, 79.4 AI/block):

$$\text{Block Reward Value} = 79.4 \times \$0.10 = \$7.94 \tag{14}$$
$$\text{Blocks per hour} = 1,800 \text{ (2-sec blocks)} \tag{15}$$
$$\text{Hourly network reward} = 1,800 \times \$7.94 = \$14,292 \tag{16}$$

If 1,000 GPUs mining:

$$\text{Per-GPU hourly} = \frac{\$14,292}{1,000} = \$14.29/\text{hr} \tag{17}$$

vs GPU operating cost: \$1.50–2.50/hr ⇒ 5–10x profitable

2.8.2 Miner Incentive Curve

Network GPUs	Per-GPU \$/hr	vs Cost	Profitable?
100	\$142.92	57x	✓✓✓
1,000	\$14.29	5.7x	✓✓
10,000	\$1.43	0.57x	Marginal
50,000	\$0.29	0.12x	Fee-dependent

Equilibrium: Network grows until per-GPU rewards ≈ operating costs.

2.9 Compute Market Economics

2.9.1 AI Service Pricing

1 AI token = 1 GPU-hour at market equilibrium:

Service	GPU-hours	AI Cost	USD (at $2)
LLM inference (1M tokens)	0.01	0.01 AI	$0.02
Image generation (100)	0.1	0.1 AI	$0.20
Fine-tuning (7B model)	10	10 AI	$20
Training (70B model)	10,000	10K AI	$20,000

2.9.2 Confidential Compute Premium

With NVTrust attestation, providers can charge premium:

Trust Level	Multiplier	Use Case
Public (no TEE)	1.0x	General inference
Private (SGX)	1.2x	Sensitive data
Confidential (H100 CC)	1.5x	Enterprise
Sovereign (Blackwell)	2.0x	Government/Finance

2.10 Long-term Contracts

2.10.1 Prepaid Compute Contracts

Smart contract for committed compute:

```
contract ComputeContract {
    struct Commitment {
        address user;
        uint256 aiAmount;      // Prepaid AI tokens
        uint256 gpuHours;      // Committed hours
        uint256 pricePerHour;  // Locked rate
        uint256 expiry;        // Contract end
        address provider;      // Assigned provider
    }

    // Lock AI tokens for 1-year commitment
    // Get 20% discount vs spot
    function createYearContract(
        uint256 monthlyHours,
        address preferredProvider
    ) external {
        uint256 totalHours = monthlyHours * 12;
        uint256 discountedPrice = spotPrice * 80 / 100;
        uint256 totalCost = totalHours * discountedPrice;

        aiToken.transferFrom(msg.sender, address(this), totalCost);

        commitments[msg.sender] = Commitment({
            user: msg.sender,
            aiAmount: totalCost,
            gpuHours: totalHours,
            pricePerHour: discountedPrice,
            expiry: block.timestamp + 365 days,
            provider: preferredProvider
        });
    }
}
```

2.10.2 Contract Tiers

Commitment	Discount	Min Hours	Lock Period
Spot	0%	0	None
Monthly	10%	100 hrs	30 days
Quarterly	15%	500 hrs	90 days
Annual	20%	2,000 hrs	365 days
Enterprise	30%	10,000 hrs	Custom

2.11 LP Mining Rewards

2.11.1 Liquidity Provider Incentives

LP token holders receive share of:

1. **Swap Fees**: 0.3% of all trades
2. **Mining Rewards**: 10% of block rewards
3. **Protocol Fees**: 5% of compute payments

$$\text{LP APY} = \frac{\text{Fees} + \text{Mining Share}}{\text{LP Value}} \times 100\% \tag{18}$$

Expected APY at launch: **50–200%** (declining as TVL grows)

2.11.2 Community Ownership

- Anyone can provide liquidity
- LP tokens = ownership share of AI ecosystem
- Mining rewards distributed proportionally
- Governance rights via LP token staking

2.12 Treasury Allocation

$$\text{Treasury} = 2\% \times \text{Mining Rewards} \tag{19}$$

Per-chain treasury (over full emission):

$$\text{Treasury Total} = 0.02 \times 1B = 20M \text{ AI per chain} \tag{20}$$

Treasury Uses:

- Protocol development (40%)
- Security audits (20%)
- Community grants (20%)
- Bridge security reserves (20%)

2.13 Cross-Chain Dynamics

2.13.1 Teleport Arbitrage

If AI trades at different prices across chains:

$$\text{Profit} = P_{\text{high}} - P_{\text{low}} - \text{bridge\_fee} \tag{21}$$

Arbitrageurs will:

1. Buy AI on low-price chain
2. Teleport to high-price chain
3. Sell for profit
4. Prices converge across chains

2.13.2 Per-Chain Supply Elasticity

Via Teleport, individual chain supplies can:

- Rise above 1B (inflows from other chains)
- Fall below 1B (outflows to other chains)
- Global supply remains fixed at $n \times 1B$

2.14 Adding New Chains

2.14.1 Governance Process

1. **Proposal**: Submit chain addition proposal
2. **Discussion**: 7-day community discussion
3. **Vote**: Token holders vote (50%+ approval)
4. **Deployment**: Auto-deploy contracts via factory
5. **Bridge**: Add to Teleport MPC config
6. **LP Seeding**: Seed initial liquidity pool

2.14.2 Self-Deployment Option

Anyone can deploy AI token to a new chain:

```
// Deploy AIToken to new chain
AITokenFactory.deploy(
    safe,           // MPC wallet address
    treasury,       // Treasury address
    teleportBridge  // Teleport bridge address
);

// Must be added to Teleport for cross-chain
// Requires governance vote or MPC approval
```

2.15 Summary: Bitcoin for AI Economics

Parameter	**Value**
Supply per chain	1,000,000,000 AI
Launch chains	10 (10B total)
LP depth per chain	$10M
Launch price	$0.10/AI
Equilibrium price	$2.00–2.50/AI
Initial block reward	79.4 AI
Halving interval	6.3M blocks (4 years)
Block time	2 seconds
Treasury rate	2%
LP mining share	10%

Value Proposition:

- Users: 25x cheaper compute at launch
- Miners: 5–10x profitable at launch
- LPs: 50–200% APY at launch
- Community: Own their AI infrastructure

3 Proof of AI Consensus

3.1 Overview

Proof of AI (PoAI) is a consensus mechanism that rewards *useful* AI compute rather than wasteful hash computation. Miners earn AI tokens by:

1. Providing inference services (LLM, vision, audio)
2. Contributing training compute for model improvement
3. Running research workloads for scientific discovery

3.2 Work Types

Definition 3.1 (AI Work Types). *PoAI recognizes three categories of useful work:*

$$W_{inference} = tokens\_processed \times model\_complexity \tag{22}$$
$$W_{training} = flops \times batch\_size \times gradient\_steps \tag{23}$$
$$W_{research} = compute\_hours \times job\_complexity \tag{24}$$

3.3 Reward Calculation

The reward for a unit of AI work is:

$$R = R_{\text{base}} \times W \times D^{-1} \times M_{\text{GPU}} \times M_{\text{uptime}} \tag{25}$$

where:

- R_{base} = base reward per compute unit
- W = work units (tokens, FLOPs, etc.)

- D = current network difficulty
- M_{GPU} = GPU tier multiplier
- M_{uptime} = uptime bonus (0.9–1.1)

3.3.1 GPU Tier Multipliers

GPU Model	NVTrust	Trust Score	Multiplier
GB200	Full + TEE-I/O	100	1.5
B200	Full + TEE-I/O	100	1.5
B100	Full + TEE-I/O	100	1.5
H200	Full	95	1.3
H100	Full	95	1.3
RTX PRO 6000	Basic	85	1.1
RTX 5090	Software only	60	0.8
RTX 4090	Software only	60	0.8

3.4 Work Proof Structure

Each mining proof contains:

```
pub struct AIWorkProof {
    // Identity
    pub miner_pubkey: [u8; 1952],      // ML-DSA-65 public key
    pub device_id: [u8; 32],           // GPU hardware ID

    // Chain binding
    pub chain_id: u64,                 // Target chain
        (36963/200200/96369)
    pub nonce: [u8; 32],               // Unique per job
    pub timestamp: u64,                // Unix timestamp

    // Work specification
    pub model_hash: [u8; 32],          // BLAKE3 of model weights
    pub input_hash: [u8; 32],          // BLAKE3 of input data
    pub output_hash: [u8; 32],         // BLAKE3 of output

    // Work metrics
    pub work_type: WorkType,
    pub compute_units: u64,
    pub tokens_processed: u64,
    pub flops: u64,

    // Attestation
    pub nvtrust_signature: Vec<u8>,    // NVTrust enclave signature
    pub spdm_evidence: Vec<u8>,        // GPU firmware measurement

    // Miner signature
    pub signature: Vec<u8>,            // ML-DSA signature over proof
}
```

3.5 Validation Rules

A proof is valid if and only if:

1. **Signature Valid**: ML-DSA signature verifies against miner pubkey
2. **NVTrust Valid**: NVTrust signature chains to NVIDIA root CA
3. **Chain Bound**: proof.chain_id = current_chain_id
4. **Not Spent**: hash(device_id‖nonce‖chain_id) $\notin$ SpentSet
5. **Fresh**: |timestamp − block_time| < PROOF_EXPIRY
6. **Meets Difficulty**: Work exceeds current difficulty target

Algorithm 1 ValidateAIWorkProof

```
Require: AIWorkProof $P$, ChainState $S$
Ensure: Boolean validity
 1: verify $P$.signature against $P$.miner_pubkey
 2: verify $P$.nvtrust_signature against NVIDIA root
 3: if $P$.chain_id $\neq$ $S$.chain_id then
 4:     return false                                        ▷ Wrong chain binding
 5: end if
 6: work_id $\leftarrow$ BLAKE3($P$.device_id‖$P$.nonce‖$P$.chain_id)
 7: if work_id $\in$ $S$.spent_set then
 8:     return false                                        ▷ Double-spend attempt
 9: end if
10: if |$P$.timestamp − $S$.block_time| > PROOF_EXPIRY then
11:     return false                                        ▷ Proof expired
12: end if
13: verify $P$.compute_units $\geq$ $S$.difficulty_target
14: return true
```

3.6 Block Structure

Blocks contain aggregated AI work proofs:

```rust
pub struct AIBlock {
    // Header
    pub height: u64,
    pub prev_hash: [u8; 32],
    pub merkle_root: [u8; 32],        // Merkle root of proofs
    pub state_root: [u8; 32],         // State trie root
    pub timestamp: u64,
    pub difficulty: u64,

    // Proofs
    pub proofs: Vec<AIWorkProof>,     // Up to MAX_PROOFS_PER_BLOCK
    pub total_work: u64,              // Sum of all work units
    pub total_rewards: u128,          // Sum of all rewards

    // Consensus
    pub proposer: [u8; 1952],         // Block proposer pubkey
    pub quasar_signature: Vec<u8>,    // Quasar consensus signature
}
```

3.7 Integration with Quasar Consensus

PoAI integrates with Quasar for instant finality:

1. **Proposer Selection**: Weighted by staked AI tokens + cumulative work
2. **Finality**: 2-round BFT with $\frac{2}{3}$ supermajority
3. **Block Time**: 500ms target (sub-second user experience)
4. **Signatures**: ML-DSA for quantum safety

Property 3.2 (Finality Guarantee)**.** *Under Quasar consensus with n validators and at most $f < n/3$ Byzantine:*

- ***Safety**: No two honest validators finalize conflicting blocks*
- ***Liveness**: If $\geq 2n/3$ validators are honest, blocks finalize within 2 rounds*

4 NVTrust Chain-Binding

4.1 Double-Spend Prevention

The fundamental security challenge in multi-chain AI mining is preventing "copy-paste mining"—submitting the same work to multiple chains for multiple rewards. PoAI solves this via **chain-binding**: cryptographically committing each unit of work to a specific chain before computation begins.

Definition 4.1 (Double-Spend Attack)**.** *An attacker performs work W once but claims rewards on chains $C_1, C_2, \ldots, C_n$ by submitting the same or slightly modified proofs.*

Theorem 4.2 (Double-Spend Resistance)**.** *Under chain-binding with NVTrust attestation, the probability of a successful double-spend is:*

$$P(\textit{double-spend}) \leq \textit{negl}(\lambda) \tag{26}$$

where λ is the security parameter (256-bit for BLAKE3).

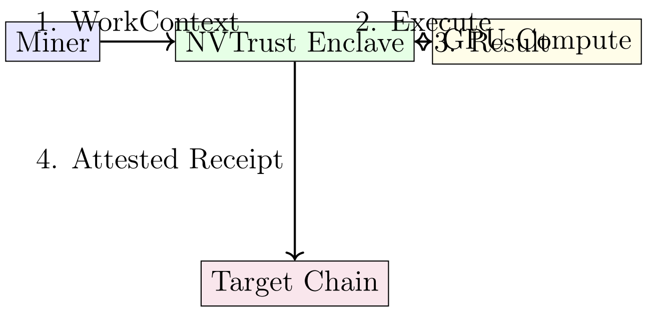
4.2 Work Context Structure

Before any computation begins, the miner commits to a **WorkContext**:

```rust
pub struct WorkContext {
    pub chain_id: u64,            // Target chain: 36963/200200/96369
    pub job_id: [u8; 32],         // Unique job identifier
    pub model_hash: [u8; 32],     // BLAKE3(model_weights)
    pub input_hash: [u8; 32],     // BLAKE3(input_data)
    pub device_id: [u8; 32],      // GPU hardware identifier
    pub nonce: [u8; 32],          // Fresh randomness
    pub timestamp: u64,           // Unix timestamp
}
```

4.3 NVTrust Attestation Flow

1. **Pre-Compute Commitment**: Miner creates WorkContext with target chain ID
2. **TEE Initialization**: Context passed to NVTrust enclave
3. **Secure Execution**: AI workload runs inside GPU TEE
4. **Receipt Generation**: Enclave creates attested receipt
5. **Hardware Signature**: NVTrust signs receipt with device key
6. **Chain Submission**: Receipt submitted to committed chain



4.4 Attested Receipt

The NVTrust enclave produces an **AttestedReceipt**:

```
pub struct AttestedReceipt {
    pub context: WorkContext,         // Original commitment
    pub result_hash: [u8; 32],        // BLAKE3(output)
    pub work_metrics: WorkMetrics,    // FLOPs, tokens, time
    pub nvtrust_signature: Vec<u8>,   // GPU hardware signature
    pub spdm_evidence: SPDMEvidence,  // Firmware measurements
}

pub struct SPDMEvidence {
    pub version: u8,
    pub measurement_hash: [u8; 48],   // GPU firmware hash
    pub nonce: [u8; 32],              // Replay protection
    pub signature: Vec<u8>,           // SPDM signature
    pub certificate_chain: Vec<u8>,   // Chain to NVIDIA root
}
```

4.5 Spent Set

Each chain maintains a **SpentSet** to track minted work:

Definition 4.3 (Work ID). *The unique identifier for a unit of work is:*

$$work\_id = BLAKE3(device\_id \| nonce \| chain\_id) \tag{27}$$

Property 4.4 (Spent Set Invariant). *For all valid blocks B and proofs $P \in B$:*

$$work\_id(P) \notin SpentSet(B.prev) \wedge work\_id(P) \in SpentSet(B) \tag{28}$$

Algorithm 2 CheckAndMarkSpent

```
Require: AttestedReceipt $R$, SpentSet $S$
Ensure: Boolean (success), SpentSet (updated)
1: work_id ← BLAKE3($R$.context.device_id$\|$$R$.context.nonce$\|$$R$.context.chain_id)
2: if work_id $\in S$ then
3:     return (false, $S$)                                  ▷ Already minted
4: end if
5: $S' \leftarrow S \cup \{$work_id$\}$
6: return (true, $S'$)
```

4.6 Chain ID Validation

Before processing any receipt, validators check chain binding:

```rust
fn validate_chain_binding(
    receipt: &AttestedReceipt,
    expected_chain_id: u64,
) -> Result<(), MiningError> {
    if receipt.context.chain_id != expected_chain_id {
        return Err(MiningError::WrongChain {
            expected: expected_chain_id,
            got: receipt.context.chain_id,
        });
    }
    Ok(())
}
```

4.7 Multi-Chain Mining

The same GPU can mine for multiple chains, but requires **separate work** for each:

GPU	Hanzo (36963)	Zoo (200200)	Lux (96369)
H100-001	nonce: 0x1a...	nonce: 0x2b...	nonce: 0x3c...
H100-001	Receipt A	Receipt B	Receipt C
H100-001	✓Valid	✓Valid	✓Valid

Lemma 4.5 (Cross-Chain Independence). *Receipts R_A and R_B with different chain IDs are independent:*

$$work\_id(R_A) \neq work\_id(R_B) \iff R_A.chain\_id \neq R_B.chain\_id \tag{29}$$

4.8 Supported GPU Hardware

GPU Model	CC Support	Trust Score	Reward Multiplier
GB200	Full NVTrust + TEE-I/O	100	1.5x
B200	Full NVTrust + TEE-I/O	100	1.5x
B100	Full NVTrust + TEE-I/O	100	1.5x
H200	Full NVTrust	95	1.3x
H100	Full NVTrust	95	1.3x
RTX PRO 6000	NVTrust	85	1.1x
RTX 5090	Software only	60	0.8x
RTX 4090	Software only	60	0.8x

Key Invariant: The same AI work cannot be minted on Hanzo, Lux, AND Zoo—only on the chain specified in the pre-committed WorkContext.chain_id.

5 Difficulty Adjustment

5.1 Overview

Unlike Bitcoin's hash-based difficulty, PoAI uses a **compute-based difficulty** that adjusts based on network AI throughput. The goal is to maintain consistent block times while adapting to changes in total compute capacity.

5.2 Difficulty Target

Definition 5.1 (Difficulty Target)**.** *The difficulty target D represents the minimum compute units required for a valid proof:*

$$D = base\_compute \times 2^{difficulty\_bits} \tag{30}$$

A proof is valid only if:

$$W_{\text{proof}} \geq D_{\text{current}} \tag{31}$$

5.3 Adjustment Algorithm

Difficulty adjusts every $N_{\text{adj}} = 2016$ blocks (similar to Bitcoin):

Algorithm 3 AdjustDifficulty

```
Require: Previous difficulty $D_{\text{prev}}$, actual time $T_{\text{actual}}$, target time $T_{\text{target}}$
Ensure: New difficulty $D_{\text{new}}$
1: ratio ← $T_{\text{actual}}/T_{\text{target}}$
2: ratio ← $\max(0.25, \min(4.0, \text{ratio}))$        ▷ Clamp to 4x range
3: $D_{\text{new}}$ ← $D_{\text{prev}}/\text{ratio}$
4: return $D_{\text{new}}$
```

5.4 Per-Chain Difficulty

Each chain maintains independent difficulty:

Chain	Target Block Time	Adjustment Window	Max Adjustment
Hanzo (36963)	2 seconds	2016 blocks	4x
Zoo (200200)	2 seconds	2016 blocks	4x
Lux (96369)	2 seconds	2016 blocks	4x

5.5 Compute Unit Standardization

Different AI workloads produce different compute metrics. We standardize to **AI Compute Units (ACU)**:

$$\text{ACU} = \alpha \times \text{tokens} + \beta \times \text{FLOPs} + \gamma \times \text{compute\_hours} \tag{32}$$

where:

- $\alpha = 10^{-3}$ (tokens to ACU)
- $\beta = 10^{-15}$ (FLOPs to ACU)
- $\gamma = 10^{6}$ (hours to ACU)

5.6 GPU Tier Adjustment

Higher-tier GPUs earn bonus difficulty credits:

$$D_{\text{effective}} = D_{\text{raw}} \times M_{\text{tier}} \tag{33}$$

GPU Tier	M_{tier}	Effective Difficulty
Sovereign (GB200, B200)	1.5	$D \times 1.5$
DataCenter (H100, H200)	1.3	$D \times 1.3$
Professional (RTX PRO)	1.1	$D \times 1.1$
Consumer (RTX 4090/5090)	0.8	$D \times 0.8$

5.7 Emergency Difficulty Adjustment

If blocks are too slow (>10x target), emergency adjustment triggers:

```
fn emergency_adjust(
    current_difficulty: u64,
    time_since_last_block: Duration,
    target_block_time: Duration,
) -> u64 {
    if time_since_last_block > target_block_time * 10 {
        // Emergency: reduce difficulty by 25%
        current_difficulty * 75 / 100
    } else {
        current_difficulty
    }
}
```

5.8 Difficulty and Rewards

Rewards scale inversely with difficulty:

$$R = R_{\text{base}} \times \frac{D_0}{D_{\text{current}}} \tag{34}$$

This ensures:

- High difficulty $\Rightarrow$ fewer proofs valid $\Rightarrow$ higher reward per proof
- Low difficulty $\Rightarrow$ more proofs valid $\Rightarrow$ lower reward per proof

5.9 Economic Equilibrium

Theorem 5.2 (Mining Equilibrium). *In equilibrium, miners earn expected value equal to their electricity and hardware costs:*

$$\mathbb{E}[Revenue] = Cost_{electricity} + Cost_{depreciation} \tag{35}$$

Difficulty auto-adjusts to maintain this equilibrium as hashrate changes.

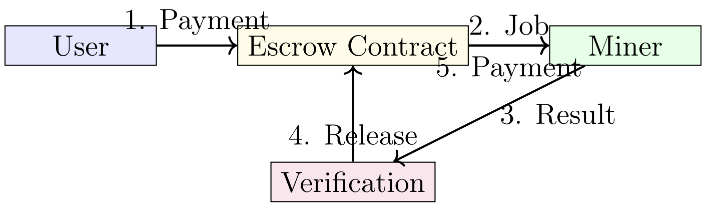
6 Market Dynamics and Payment

6.1 Overview

PoAI creates a two-sided market where:

- **Miners** provide AI compute and earn AI tokens
- **Users** pay AI tokens for AI services (inference, training, etc.)

6.2 Payment Flow



6.3 Pricing Models

Miners can offer multiple pricing models:

Model	Unit	Use Case
Per Token	AI/token	LLM inference
Per Inference	AI/request	Image generation
Per Minute	AI/minute	Real-time audio
Per FLOP	AI/TFLOP	Training
Hybrid	Mix	Custom workloads

6.4 Market-Driven Pricing

Prices adjust based on supply and demand using **Hamiltonian market dynamics**:

Definition 6.1 (Hamiltonian Market)**.** *The market state is described by position q (price) and momentum p (order flow):*

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = p \tag{36}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} - \gamma p + \xi(t) \tag{37}$$

where $H = \frac{p^2}{2} + V(q)$ is the Hamiltonian, γ is damping, and $\xi(t)$ is noise.

6.4.1 Price Update Algorithm

6.5 Escrow Contract

Payments are held in escrow until work is verified:

```
contract ComputeEscrow {
    struct Request {
        address user;
        address miner;
        uint256 payment;
        bytes32 inputHash;
```

Algorithm 4 UpdateMarketPrice

```
Require: Current price $P$, supply $S$, demand $D$, target utilization $U_{\text{target}}$
Ensure: New price $P'$
1: $U_{\text{current}} \leftarrow D/S$                          ▷ Current utilization
2: imbalance $\leftarrow U_{\text{current}} - U_{\text{target}}$
3: adjustment $\leftarrow$ imbalance $\times$ sensitivity
4: adjustment $\leftarrow$ adjustment $\times$ damping          ▷ Prevent oscillation
5: $P' \leftarrow P \times (1 + \text{adjustment})$
6: $P' \leftarrow \max(P_{\min}, \min(P_{\max}, P'))$           ▷ Bounds
7: return $P'$
```

```
        bytes32 resultHash;
        uint256 deadline;
        RequestStatus status;
    }

    function createRequest(
        bytes32 modelId,
        bytes32 inputHash,
        uint256 maxPayment,
        uint256 deadline
    ) external returns (bytes32 requestId);

    function acceptRequest(bytes32 requestId) external;

    function submitResult(
        bytes32 requestId,
        bytes32 resultHash
    ) external;

    function verifyAndRelease(bytes32 requestId) external;

    function slashAndRefund(bytes32 requestId) external;
}
```

6.6 Fee Structure

Fee Type	Rate	Recipient
Market Fee	1%	Protocol Treasury
Network Fee	Gas	Validators
Slashing	10%	User (refund)

6.7 Provider Registration

Miners register as providers with stake:

```
function registerProvider(
    uint256 stake,          // Minimum 1000 AI
    bytes32 modelId,        // Supported model
    bytes32 gpuId,          // Hardware ID
    PricingModel memory pricing,
    uint256 maxConcurrent   // Max parallel jobs
) external;
```

6.8 Slashing Conditions

Providers are slashed (10% of stake) for:

1. **Invalid Result**: Output doesn't match input/model
2. **Timeout**: No result by deadline
3. **Attestation Failure**: NVTrust verification fails

6.9 Economic Incentives

Theorem 6.2 (Incentive Compatibility). *Under the escrow mechanism, honest behavior is a Nash equilibrium:*

- *Miners: Providing valid compute maximizes expected profit*
- *Users: Paying for services maximizes utility*
- *Slashing cost exceeds fraud profit*

6.10 Per-Chain Fee Customization

Each chain can set its own fee parameters:

Parameter	Lux	Hanzo	Zoo
Base Price (AI/token)	0.0001	0.0001	0.0001
Market Fee	1%	1%	2% (research)
Min Provider Stake	1000 AI	1000 AI	500 AI
Slashing Rate	10%	10%	15%

This allows each chain's community to optimize for their specific use cases.

7 Multi-Chain Mining with Teleport

7.1 Overview

PoAI enables seamless cross-chain operations via the **Teleport Protocol**. Miners can earn on any chain, and tokens can flow freely across the ecosystem.

7.2 Supported Chains

Chain	Chain ID	Token	Role
Hanzo L1	36963	AI (native)	Mining Origin
Hanzo EVM	36963	AI, ZOO	DeFi Hub
Zoo EVM	200200	AI, ZOO	Research Focus
Lux C-Chain	96369	AI, ZOO, LUX	General Purpose

7.3 Global Supply Dynamics

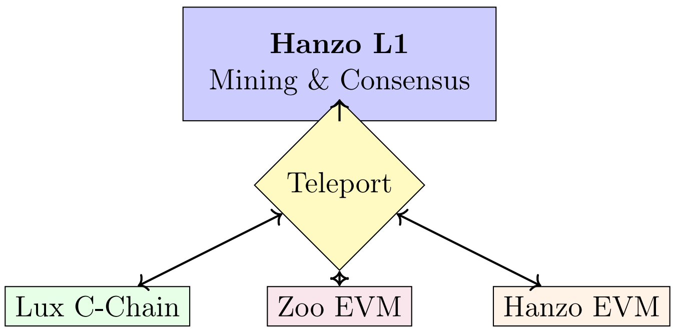
Property 7.1 (Expanding Global Cap). *The global AI supply cap increases only when new chains join:*

$$S_{global} = n \times 10^9 \ AI \tag{38}$$

where n is the number of active chains. This creates predictable scarcity while allowing ecosystem growth.

7.4 Teleport Protocol

7.4.1 Architecture



7.4.2 Transfer Flow

1. **Lock/Burn**: Source chain locks or burns tokens
2. **MPC Validation**: Top 100 validators verify the transfer
3. **Threshold Signature**: CGGMP20 produces collective signature
4. **Mint/Release**: Destination chain mints or releases tokens

7.4.3 Transfer Message

```
pub struct TeleportTransfer {
    pub teleport_id: [u8; 32],         // Unique transfer ID
    pub source_chain: u64,             // Origin chain ID
    pub destination_chain: u64,        // Target chain ID
    pub sender: Vec<u8>,               // ML-DSA public key
    pub recipient: String,             // Destination address
    pub amount: u128,                  // AI tokens (neurons)
    pub nonce: u64,                    // Replay protection
    pub timestamp: u64,                // Unix timestamp
    pub signature: Vec<u8>,            // ML-DSA signature
}
```

7.5 Mining on Multiple Chains

A single GPU can mine for all chains simultaneously by:

1. Creating separate WorkContext for each chain
2. Generating unique nonces per chain
3. Submitting receipts to respective chains

```
// Mine for all three chains in parallel
let hanzo_ctx = WorkContext::for_chain(ChainId::Hanzo, &job);
let zoo_ctx = WorkContext::for_chain(ChainId::Zoo, &job);
let lux_ctx = WorkContext::for_chain(ChainId::Lux, &job);

// Execute and submit
let hanzo_receipt = execute_and_attest(hanzo_ctx).await?;
let zoo_receipt = execute_and_attest(zoo_ctx).await?;
let lux_receipt = execute_and_attest(lux_ctx).await?;

submit_to_chain(ChainId::Hanzo, hanzo_receipt).await?;
submit_to_chain(ChainId::Zoo, zoo_receipt).await?;
submit_to_chain(ChainId::Lux, lux_receipt).await?;
```

Note: Each submission requires *separate work*—the same work cannot be submitted to multiple chains.

7.6 Cross-Chain Reward Claiming

Miners can claim rewards on any chain:

```
interface IRewardClaiming {
    // Claim reward mined on current chain
    function claimLocalReward(
        bytes calldata proof,
        bytes calldata signature
    ) external returns (uint256);

    // Claim reward mined on another chain via Teleport
    function claimCrossChainReward(
        bytes32 teleportId,
        bytes32[] calldata merkleProof
    ) external returns (uint256);
}
```

7.7 Liquidity Pools

Cross-chain liquidity enables efficient markets:

Pool	Chain	Purpose
AI/ETH	Lux C-Chain	External liquidity
AI/ZOO	Zoo EVM	Governance pairing
AI/LUX	All	Native pairing
AI/USDC	All	Stable pairing

7.8 Chain-Specific Features

7.8.1 Lux C-Chain (96369)

- General-purpose EVM
- Highest liquidity
- Gateway to external ecosystems

7.8.2 Hanzo EVM (36963)

- AI-focused applications
- Native mining origin
- Lowest latency for AI services

7.8.3 Zoo EVM (200200)

- Research and DeSci focus
- Higher treasury allocation (2%)
- Governance over research grants

7.9 Adding New Chains

New chains can join the ecosystem by:

1. **Governance Proposal**: Submit proposal to add chain
2. **Technical Integration**: Deploy Teleport contracts
3. **Validator Opt-In**: MPC nodes monitor new chain
4. **Supply Allocation**: 1B AI cap assigned to new chain

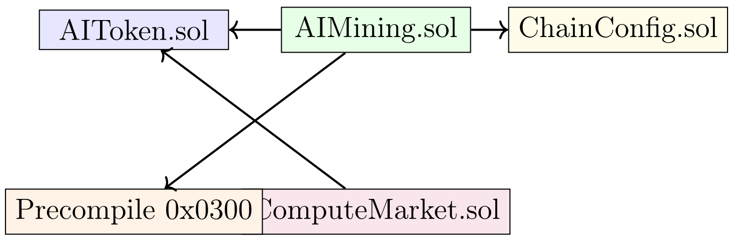
$$S_{\text{global}}^{\text{new}} = S_{\text{global}}^{\text{old}} + 10^9 \tag{39}$$

8 EVM Contracts

8.1 Overview

PoAI deploys a suite of smart contracts on each EVM chain for mining reward management, token operations, and market functionality.

8.2 Contract Architecture



8.3 AIToken.sol

ERC20 token with mining-based minting:

```
contract AIToken is ERC20, AccessControl {
    uint256 public constant MAX_SUPPLY = 1_000_000_000 * 1e18;
    uint256 public constant HALVING_INTERVAL = 210_000;
    uint256 public constant INITIAL_REWARD = 50 * 1e18;
    uint256 public constant TREASURY_RATE = 200; // 2%

    uint256 public totalMinted;
    uint256 public genesisBlock;
    address public treasury;

    function mintReward(
        address miner,
        uint256 amount
    ) external onlyRole(MINER_ROLE) {
        require(totalMinted + amount <= MAX_SUPPLY, "Supply␣cap");

        uint256 treasuryAmount = amount * TREASURY_RATE / 10000;
        uint256 minerAmount = amount - treasuryAmount;

        _mint(miner, minerAmount);
        _mint(treasury, treasuryAmount);
        totalMinted += amount;

        emit RewardMinted(miner, minerAmount, currentEpoch());
    }

    function currentEpoch() public view returns (uint256) {
        return (block.number - genesisBlock) / HALVING_INTERVAL;
    }

    function currentReward() public view returns (uint256) {
        uint256 epoch = currentEpoch();
        return INITIAL_REWARD >> epoch; // Halving
    }
}
```

8.4 AIMining.sol

Core mining contract for proof submission:

```
contract AIMining {
    struct WorkProof {
        bytes32 sessionId;
        bytes32 nonce;
        bytes32 gpuId;
        bytes32 computeHash;
        uint256 timestamp;
        GPUTier tier;
        bytes signature;
    }

    mapping(bytes32 => bool) public spentProofs;
    IChainConfig public config;
    IAIToken public token;

    function submitProof(
        WorkProof calldata proof
    ) external returns (uint256 reward) {
        // 1. Compute work ID
        bytes32 workId = keccak256(abi.encodePacked(
            proof.gpuId,
            proof.nonce,
            block.chainid
        ));

        // 2. Check spent set
        require(!spentProofs[workId], "Already␣minted");

        // 3. Verify via precompile
        require(
            IAIMiningPrecompile(PRECOMPILE).verifyMLDSA(
                proof.gpuId,
                abi.encode(proof),
                proof.signature
            ),
            "Invalid␣signature"
        );

        // 4. Calculate reward
        reward = calculateReward(proof);

        // 5. Mark spent and mint
        spentProofs[workId] = true;
        token.mintReward(msg.sender, reward);

        emit ProofSubmitted(msg.sender, workId, reward);
    }

    function calculateReward(
        WorkProof calldata proof
    ) public view returns (uint256) {
        uint256 baseReward = token.currentReward();
        uint256 multiplier = config.getGPUMultiplier(proof.tier);
        return baseReward * multiplier / 10000;
    }
}
```

8.5 ChainConfig.sol

Per-chain configuration management:

```
contract ChainConfig {
    struct Config {
        uint256 baseReward;
        uint256 halvingInterval;
        uint256 difficultyTarget;
        uint256 treasuryRate;
        mapping(GPUTier => uint256) gpuMultipliers;
    }

    mapping(uint256 => Config) public chainConfigs;

    function initializeChain(
        uint256 chainId,
        uint256 baseReward,
        uint256 treasuryRate
    ) external onlyAdmin {
        Config storage cfg = chainConfigs[chainId];
        cfg.baseReward = baseReward;
        cfg.halvingInterval = 210_000;
        cfg.treasuryRate = treasuryRate;

        // Default GPU multipliers
        cfg.gpuMultipliers[GPUTier.Consumer] = 8000;      // 0.8x
        cfg.gpuMultipliers[GPUTier.Professional] = 11000; // 1.1x
        cfg.gpuMultipliers[GPUTier.DataCenter] = 13000;   // 1.3x
        cfg.gpuMultipliers[GPUTier.Sovereign] = 15000;    // 1.5x
    }

    function getGPUMultiplier(
        GPUTier tier
    ) external view returns (uint256) {
        return chainConfigs[block.chainid].gpuMultipliers[tier];
    }
}
```

8.6 ComputeMarket.sol

Decentralized AI service marketplace:

```
contract ComputeMarket {
    struct Provider {
        uint256 stake;
        bytes32 modelId;
        bytes32 gpuId;
        PricingModel pricing;
        uint256 activeRequests;
        uint256 maxConcurrent;
        uint256 reputation;
    }

    struct Request {
        address user;
        address provider;
        bytes32 modelId;
        bytes32 inputHash;
        bytes32 resultHash;
        uint256 payment;
        uint256 deadline;
        RequestStatus status;
    }

    mapping(address => Provider) public providers;
    mapping(bytes32 => Request) public requests;

    function createRequest(
        bytes32 modelId,
        bytes32 inputHash,
        uint256 maxPayment,
        uint256 duration
    ) external returns (bytes32 requestId) {
        // Transfer payment to escrow
        token.transferFrom(msg.sender, address(this), maxPayment);

        requestId = keccak256(abi.encodePacked(
            msg.sender,
            modelId,
            block.timestamp
        ));

        requests[requestId] = Request({
            user: msg.sender,
            provider: address(0),
            modelId: modelId,
            inputHash: inputHash,
            resultHash: bytes32(0),
            payment: maxPayment,
            deadline: block.timestamp + duration,
            status: RequestStatus.Open
        });
    }

    function submitResult(
        bytes32 requestId,
        bytes32 resultHash
    ) external {
        Request storage req = requests[requestId];
        require(req.provider == msg.sender, "Not␣provider");
        require(req.status == RequestStatus.Accepted, "Not␣accepted");

        req.resultHash = resultHash;
        req.status = RequestStatus.Completed;

        // Release payment minus market fee
        uint256 fee = req.payment * MARKET_FEE / 10000;
        token.transfer(msg.sender, req.payment - fee);
        token.transfer(treasury, fee);
    }
}
```

8.7 Precompile Interface

High-performance native operations at address 0x0300:

```
interface IAIMiningPrecompile {
    /// @notice Verify ML-DSA quantum-safe signature
    /// @param pubkey ML-DSA public key (1952 bytes for Level 3)
    /// @param message Message that was signed
    /// @param signature ML-DSA signature
    /// @return valid True if signature is valid
    function verifyMLDSA(
        bytes calldata pubkey,
        bytes calldata message,
        bytes calldata signature
    ) external view returns (bool valid);

    /// @notice Calculate reward for work proof
    /// @param workProof Encoded work proof
    /// @param chainId Target chain ID
    /// @return reward Calculated reward in neurons
    function calculateReward(
        bytes calldata workProof,
        uint64 chainId
    ) external view returns (uint256 reward);

    /// @notice Verify NVTrust GPU attestation
    /// @param receipt Attested receipt from NVTrust
    /// @param signature NVTrust signature
    /// @return valid True if attestation is valid
    function verifyNVTrust(
        bytes calldata receipt,
        bytes calldata signature
    ) external view returns (bool valid);

    /// @notice Check if work ID is in spent set
    /// @param workId Work identifier
    /// @return spent True if already minted
    function isSpent(bytes32 workId) external view returns (bool spent)
        ;

    /// @notice Compute work ID from components
    /// @param deviceId GPU device ID
    /// @param nonce Unique nonce
    /// @param chainId Target chain ID
    /// @return workId Computed work identifier
    function computeWorkId(
        bytes32 deviceId,
        bytes32 nonce,
        uint64 chainId
    ) external pure returns (bytes32 workId);
}
```

8.8 Gas Costs

Operation	Precompile	Pure Solidity
verifyMLDSA	3,000 gas	500,000+ gas
calculateReward	1,000 gas	10,000 gas
verifyNVTrust	5,000 gas	1,000,000+ gas
isSpent	100 gas	2,100 gas (SLOAD)
computeWorkId	50 gas	500 gas

8.9 Deployment Addresses

Contract	Address	All Chains
AIToken	0xAI00...0001	✓
AIMining	0xAI00...0002	✓
ChainConfig	0xAI00...0003	✓
ComputeMarket	0xAI00...0004	✓
Precompile	0x0300	✓

9 Security Analysis

9.1 Threat Model

We consider adversaries with the following capabilities:

1. **Malicious Miners**: Control GPU hardware, attempt double-spend
2. **Network Attackers**: Observe and manipulate network traffic
3. **Quantum Adversaries**: Future quantum computers attacking signatures
4. **Colluding Validators**: Up to $f < n/3$ Byzantine validators

9.2 Quantum Safety

9.2.1 ML-DSA Signatures

All mining signatures use ML-DSA (FIPS 204), providing NIST Level 3 security:

Algorithm	Classical	Quantum	Key Size
ECDSA (current)	128-bit	∼64-bit	33 bytes
ML-DSA-65	192-bit	128-bit	1,952 bytes

Theorem 9.1 (Quantum Resistance). *Under the Module-LWE hardness assumption, ML-DSA signatures are existentially unforgeable under chosen-message attack with quantum adversaries.*

9.2.2 Quantum Timeline

Year	Threat Level	Mitigation
2024–2026	Low	ML-DSA optional
2026–2028	Medium	ML-DSA default
2030+	High	ECDSA deprecated

9.3 Double-Spend Prevention

9.3.1 Spent Set Security

Lemma 9.2 (Spent Set Collision Resistance). *The probability of two distinct work units having the same work ID is:*

$$P(collision) \leq \frac{q^2}{2^{257}} \approx 0 \tag{40}$$

where q is the number of work units and we use 256-bit BLAKE3.

9.3.2 Chain Binding Security

Theorem 9.3 (Chain Binding Unforgeability). *An adversary cannot produce a valid receipt for chain C_2 from work performed for chain C_1:*

$$Adv_{forge}^{chain\text{-}bind}(\mathcal{A}) \leq Adv_{forge}^{NVTrust}(\mathcal{A}) + negl(\lambda) \tag{41}$$

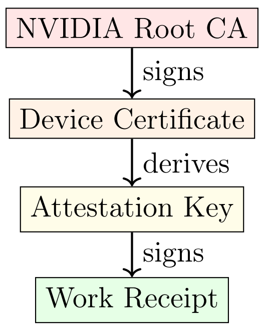
Proof. Chain ID is included in the NVTrust-signed receipt. Modifying the chain ID invalidates the signature. Creating a new valid signature requires breaking NVTrust attestation security. □

9.4 NVTrust Security

9.4.1 Trust Assumptions

1. NVIDIA root CA is trusted and not compromised
2. GPU firmware is correctly implemented
3. SPDM protocol is secure against replay and modification
4. Hardware attestation keys are protected by GPU TEE

9.4.2 Attestation Chain



9.5 Consensus Security

9.5.1 Quasar BFT Guarantees

With n validators and $f < n/3$ Byzantine:

Property 9.4 (Safety)**.** *No two honest validators finalize conflicting blocks:*

$$\forall v_1, v_2 \in \mathit{Honest} : \mathit{finalized}(v_1) \cap \mathit{finalized}(v_2) \textit{ is consistent} \tag{42}$$

Property 9.5 (Liveness)**.** *If $\geq 2n/3$ validators are honest and network is synchronous, blocks finalize:*

$$\forall tx : \exists t : \mathit{finalized}(tx) \textit{ by time } t \tag{43}$$

9.6 Teleport Security

9.6.1 MPC Security

Teleport uses CGGMP20 threshold ECDSA with t-of-n security:

Theorem 9.6 (Threshold Security)**.** *The combined private key is never reconstructed. At least t parties must collude to sign.*

9.6.2 Replay Protection

Each transfer has a unique ID:

$$\text{teleport\_id} = \text{BLAKE3}(\text{source}\|\text{dest}\|\text{sender}\|\text{amount}\|\text{nonce}) \tag{44}$$

9.7 Economic Security

9.7.1 Mining Attacks

Attack	**Cost**	**Mitigation**
Double-spend	NVTrust break	Hardware attestation
Selfish mining	Opportunity cost	Instant finality
Empty blocks	No reward	Work requirement

9.7.2 Slashing Conditions

Misbehaving providers lose stake:

Violation	Penalty
Invalid compute result	10% stake
Timeout / no response	10% stake
Attestation fraud	100% stake
Double-sign	100% stake

9.8 Key Zeroization

All secret keys are zeroized on drop:

```rust
#[derive(Zeroize, ZeroizeOnDrop)]
pub struct SecretKey(Vec<u8>);

impl Drop for SecretKey {
    fn drop(&mut self) {
        self.0.zeroize();
    }
}
```

9.9 Audit Status

Component	Auditor	Status
Smart Contracts	TBD	Pending
ML-DSA Implementation	TBD	Pending
NVTrust Integration	TBD	Pending
Teleport Protocol	TBD	Pending

10 Conclusion

10.1 Summary

Proof of AI (PoAI) establishes an open protocol for decentralized AI mining with the following key properties:

1. **Open Participation**: Anyone with GPU compute can mine AI tokens
2. **Useful Work**: Rewards actual AI computation, not wasteful hashing
3. **Bitcoin Economics**: 1B supply cap per chain, halving schedule
4. **Quantum Safety**: ML-DSA signatures protect long-term value
5. **Multi-Chain**: Seamless operation across Hanzo, Zoo, and Lux
6. **Double-Spend Proof**: NVTrust chain-binding prevents fraud
7. **Market Dynamics**: Supply/demand pricing for AI services
8. **Future Privacy**: Upgrade path to shielded mining via ZK proofs

10.2 Ecosystem Benefits

10.2.1 For Miners

- Monetize idle GPU capacity
- Earn from multiple chains simultaneously
- Higher rewards for premium hardware
- Quantum-safe earnings protection

10.2.2 For Users

- Access to decentralized AI compute
- Market-driven competitive pricing
- Cross-chain token portability
- Trustless escrow payments

10.2.3 For Developers

- Standard interfaces across all chains
- Precompiles for efficient operations
- Open-source reference implementations
- Comprehensive documentation

10.3 Comparison with Alternatives

Property	PoAI	Bitcoin	PoS	Render	Akash
Useful work	✓			✓	✓
Fixed supply	✓	✓			
Quantum-safe	✓				
Multi-chain	✓				
Hardware TEE	✓				
Instant finality	✓		✓		

10.4 Roadmap

Version	Target	Features
v1.0	Q1 2025	Public mining with NVTrust
v1.1	Q2 2025	Compute marketplace launch
v1.2	Q3 2025	Additional GPU support
v2.0	Q4 2025	Optional shielded mining (ZK)
v2.1	Q1 2026	Cross-chain atomic swaps
v3.0	Q2 2026	Default shielded mining

10.5 Future Work

10.5.1 Near-Term

1. Complete NVTrust SDK integration
2. Security audits of all contracts
3. Testnet deployment and stress testing
4. Developer documentation and tutorials

10.5.2 Medium-Term

1. ZK circuit implementation for shielded mining
2. Additional chain integrations
3. Mobile wallet support
4. Hardware wallet integration

10.5.3 Long-Term

1. Fully homomorphic compute verification
2. Decentralized model training coordination
3. AI governance via token voting
4. Integration with external AI ecosystems

10.6 Open Source

All PoAI implementations are open source:

Component	**Repository**
Protocol Specification	`github.com/luxfi/ai`
Smart Contracts	`github.com/luxfi/standard`
Mining Client	`github.com/hanzoai/node`
Precompiles	`github.com/luxfi/precompiles`
Documentation	`github.com/luxfi/ai/docs`

10.7 Acknowledgments

The PoAI protocol draws on the work of many researchers and developers:

- Satoshi Nakamoto for Bitcoin's elegant economic design
- NVIDIA for NVTrust confidential computing
- NIST for ML-DSA standardization
- The Lux, Hanzo, and Zoo communities

10.8 Closing Remarks

Proof of AI represents a fundamental shift in how we think about AI compute. By combining Bitcoin's elegant economic model with useful AI work, quantum-safe cryptography, and multi-chain interoperability, PoAI creates an open foundation for the AI economy.

"The future of AI should be open, decentralized, and owned by everyone who contributes."

Build the future. Mine AI.

11 Shielded Mining via Zero-Knowledge Proofs

11.1 Overview

While public mining (Section 3) establishes a transparent economy, many miners require privacy for competitive or regulatory reasons. **Shielded Mining** integrates with the Lux Z-Chain to enable:

1. **Hidden Miner Identity**: Device IDs and public keys masked via nullifiers
2. **Hidden Workloads**: Model, input, and output data remain private
3. **Verifiable Compute**: ZK proofs attest to valid AI work without revealing details
4. **Selective Disclosure**: Gradual reveal for auditing when required

11.2 Z-Chain Integration

The Lux Z-Chain provides the cryptographic infrastructure for shielded mining. Work proofs submitted to Z-Chain replace public data with ZK commitments.

Definition 11.1 (Shielded Work Proof)**.** *A shielded work proof* π *consists of:*

$$\pi = (\mathit{commitment}, \mathit{nullifier}, \mathit{zkproof}, \mathit{timestamp}) \tag{45}$$

where:

- $\mathit{commitment} = \mathit{Commit}(\mathit{device\_id}, \mathit{work\_data}, r)$ *for random* r
- *nullifier prevents double-spend without revealing identity*
- *zkproof attests to valid AI work*
- *timestamp binds the proof to a time window*

11.2.1 Commitment Scheme

We use Pedersen commitments over the BLS12-381 curve:

$$\text{Commit}(m, r) = m \cdot G + r \cdot H \tag{46}$$

where G and H are independent generators. This provides:

- **Hiding**: Given $\text{Commit}(m, r)$, m is computationally hidden
- **Binding**: Cannot find (m', r') such that $\text{Commit}(m, r) = \text{Commit}(m', r')$

11.3 Nullifier Scheme

The nullifier scheme prevents double-spend while hiding miner identity.

Definition 11.2 (Nullifier). *For a miner with secret key sk and work identifier work_ id:*

$$\textit{nullifier} = \mathcal{H}(sk \| work\_id) \tag{47}$$

where $\mathcal{H}$ is a collision-resistant hash function (BLAKE3).

Property 11.3 (Double-Spend Prevention). *The spent set $\mathcal{S}$ stores nullifiers, not work IDs. A proof π is rejected if:*

$$\pi.\textit{nullifier} \in \mathcal{S} \tag{48}$$

This prevents double-spend without revealing which work was performed or by whom.

11.3.1 Nullifier Properties

1. **Uniqueness**: Each (miner, work) pair produces a unique nullifier
2. **Unlinkability**: Different nullifiers from the same miner cannot be linked
3. **Non-forgery**: Cannot produce valid nullifier without knowing sk

Lemma 11.4 (Nullifier Security). *Under the collision resistance of $\mathcal{H}$:*

1. *An adversary cannot find $(sk', work\_id')$ producing the same nullifier as honest $(sk, work\_id)$*
2. *Given nullifiers n_1, n_2 from the same miner, distinguishing them from random requires breaking the PRF property of $\mathcal{H}$*

11.4 ZK Circuit Design

The ZK circuit proves: "I performed valid AI work for chain C with nonce N" without revealing device ID, model, or input/output data.

11.4.1 Circuit Statement

Definition 11.5 (ZK Statement). *Public inputs:*

$$\mathbf{x} = (chain\_id, commitment, nullifier, work\_type, compute\_units, timestamp) \tag{49}$$

Private witness:

$$\mathbf{w} = (sk, device\_id, nonce, model\_hash, input\_hash, output\_hash, r) \tag{50}$$

The circuit enforces:

$$\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1 \iff \begin{cases} \text{commitment} = \text{Commit}((\text{device\_id}, \text{model\_hash}, \text{input\_hash}, \text{output\_hash}), r) \\ \text{nullifier} = \mathcal{H}(\text{sk}\|\text{device\_id}\|\text{nonce}\|\text{chain\_id}) \\ \text{ValidWork}(\text{work\_type}, \text{compute\_units}, \text{model\_hash}) = 1 \\ \text{NVTrustVerify}(\text{device\_id}, \text{model\_hash}, \text{input\_hash}, \text{output\_hash}) = 1 \end{cases} \tag{51}$$

Algorithm 5 ShieldedWorkCircuit

```
Require: Public: (chain_id, commitment, nullifier, work_type, compute_units, timestamp)
Require: Private: (sk, device_id, nonce, model_hash, input_hash, output_hash, $r$)
Ensure: Boolean: circuit satisfiability
                                                  ▷ 1. Verify commitment opening
data ← Pack(device_id, model_hash, input_hash, output_hash)
assert commitment = PedersenCommit(data, $r$)
                                                  ▷ 2. Verify nullifier derivation
preimage ← sk||device_id||nonce||chain_id
assert nullifier = BLAKE3(preimage)
                                                  ▷ 3. Verify work validity
if work_type = INFERENCE then
    assert compute_units ≥ MIN_INFERENCE_UNITS
    assert model_hash ∈ APPROVED_MODELS
else if work_type = TRAINING then
    assert compute_units ≥ MIN_TRAINING_FLOPS
else if work_type = RESEARCH then
    assert compute_units ≥ MIN_RESEARCH_HOURS
end if
                                                  ▷ 4. Verify NVTrust attestation (in-circuit)
attestation ← NVTrustAttestation(device_id)
assert VerifySignature(attestation, NVIDIA_ROOT_PK)
assert attestation.device_id = device_id
                                                  ▷ 5. Verify timestamp freshness (relative check)
assert timestamp > 0
return true
```

11.4.2 Circuit Pseudocode

11.4.3 Proving System Selection

System	Proof Size	Verify Time	Setup
Groth16	128 bytes	3ms	Trusted
PLONK	400 bytes	8ms	Universal
Halo2	600 bytes	12ms	None

Recommendation: Use **Groth16** for production (smallest proofs, fastest verification) with a trusted setup ceremony. Use **PLONK** for development and chains requiring universal setup.

11.4.4 Circuit Constraints

The circuit requires approximately:

- **Pedersen commitment**: ∼1,500 constraints
- **BLAKE3 hash**: ∼25,000 constraints per invocation
- **NVTrust signature verification**: ∼200,000 constraints (BLS)
- **Range checks and logic**: ∼5,000 constraints

Total: ∼250,000 constraints (Groth16 proving time: ∼5 seconds on modern CPU).

11.5 Privacy Levels

Shielded mining supports multiple privacy levels, allowing miners to choose their disclosure preference.

Definition 11.6 (Privacy Levels).

$$\textit{Level 0 (Public)} : \textit{All data public} \tag{52}$$
$$\textit{Level 1 (Identity)} : \textit{Miner identity hidden} \tag{53}$$
$$\textit{Level 2 (Workload)} : \textit{Model and I/O hidden} \tag{54}$$
$$\textit{Level 3 (Full)} : \textit{All data in ZK} \tag{55}$$

Data Field	Level 0	Level 1	Level 2	Level 3
Miner pubkey	Public	Hidden	Hidden	Hidden
Device ID	Public	Hidden	Hidden	Hidden
Chain ID	Public	Public	Public	Hidden*
Model hash	Public	Public	Hidden	Hidden
Input hash	Public	Public	Hidden	Hidden
Output hash	Public	Public	Hidden	Hidden
Compute units	Public	Public	Public	Hidden*
Work type	Public	Public	Public	Hidden*

*Level 3 hides these in ZK but proves they meet minimum thresholds.

11.5.1 Level 1: Hidden Miner Identity

Only the nullifier is exposed; device ID and public key are hidden.

```rust
pub struct Level1ShieldedProof {
    // Public
    pub nullifier: [u8; 32],
    pub chain_id: u64,
    pub model_hash: [u8; 32],
    pub input_hash: [u8; 32],
    pub output_hash: [u8; 32],
    pub compute_units: u64,
    pub timestamp: u64,

    // ZK proof that hidden identity is valid
    pub zk_proof: Vec<u8>,
}
```

11.5.2 Level 2: Hidden Workload

Model and I/O data are hidden; only compute units and work type are public.

```rust
pub struct Level2ShieldedProof {
    // Public
    pub nullifier: [u8; 32],
    pub chain_id: u64,
    pub work_type: WorkType,
    pub compute_units: u64,
    pub timestamp: u64,

    // Commitments to hidden data
    pub model_commitment: [u8; 32],
    pub io_commitment: [u8; 32],

    // ZK proof
    pub zk_proof: Vec<u8>,
}
```

11.5.3 Level 3: Fully Shielded

All data hidden; ZK proof attests to minimum thresholds.

```rust
pub struct Level3ShieldedProof {
    // Only commitment and nullifier exposed
    pub commitment: [u8; 32],
    pub nullifier: [u8; 32],
    pub timestamp: u64,

    // ZK proof attesting to:
    // - Valid chain binding
    // - compute_units >= MIN_THRESHOLD
    // - work_type in {INFERENCE, TRAINING, RESEARCH}
    // - Valid NVTrust attestation
    pub zk_proof: Vec<u8>,
}
```

11.6 Selective Disclosure

Miners may need to reveal data for auditing, dispute resolution, or regulatory compliance. The protocol supports gradual disclosure.

Definition 11.7 (Opening)**.** *An opening o for commitment $c = Commit(m, r)$ is the tuple (m, r). Given o, anyone can verify:*

$$c \overset{?}{=} Commit(m, r) \tag{56}$$

11.6.1 Disclosure Levels

1. **Identity Disclosure**: Reveal $(device\_id, \mathrm{sk\_commitment})$ to prove ownership
2. **Work Disclosure**: Reveal $(model\_hash, input\_hash, output\_hash)$ to prove work details
3. **Full Disclosure**: Reveal all witness data, converting to Level 0 proof

Algorithm 6 SelectiveDisclosure

```
Require: ShieldedProof $\pi$, DisclosureRequest $D$, MinerSecret sk
Ensure: DisclosureProof $\delta$
 1: if $D$.level = IDENTITY then
 2:     $\delta \leftarrow (device\_id, \mathrm{Commit}(\mathrm{sk}, r'))$
 3:     $\delta$.proof ← ProveKnowledge(sk, $r'$)
 4: else if $D$.level = WORK then
 5:     $\delta \leftarrow (model\_hash, input\_hash, output\_hash, r)$
 6:     verify $\pi$.commitment = Commit($\delta, r$)
 7: else if $D$.level = FULL then
 8:     $\delta \leftarrow (\mathrm{sk}, device\_id, nonce, model\_hash, input\_hash, output\_hash, r)$
 9: end if
10: return $\delta$
```

11.7 Upgrade Path

The protocol evolves from fully public to default shielded over multiple versions.

Definition 11.8 (Version Timeline)**.**

$$\textit{v1.0} : \textit{Public mining with NVTrust (current)} \tag{57}$$
$$\textit{v2.0} : \textit{Optional shielded mining via Z-Chain} \tag{58}$$
$$\textit{v3.0} : \textit{Default shielded with public opt-in} \tag{59}$$

11.7.1 v1.0: Public Mining (Current)

All mining is public as described in Section 3. This establishes:

- Baseline security model with NVTrust
- Economic equilibrium and difficulty adjustment
- Infrastructure for reward distribution

11.7.2 v2.0: Optional Shielded Mining

Miners may choose shielded mode by submitting proofs to Z-Chain.

```
pub enum MiningMode {
    Public,            // Standard AIWorkProof
    Shielded(Level),   // Level1/2/3 ShieldedProof
}

pub struct MiningConfig {
    pub mode: MiningMode,
    pub chain_id: u64,
    pub zk_prover: Option<ProverConfig>,
}
```

Transition Rules:

1. Shielded proofs receive **same rewards** as public proofs
2. Spent sets are **unified**: nullifiers and work IDs share the same set
3. Shielded-to-public **conversion** allowed via full disclosure

11.7.3 v3.0: Default Shielded

Privacy becomes the default; public mining requires explicit opt-in.

- **Default**: Level 2 shielded (hidden identity and workload)
- **Opt-in Public**: Miners may choose Level 0 for transparency bonuses
- **Regulatory Mode**: Jurisdictions may require Level 1 minimum

Property 11.9 (Backward Compatibility). *All v1.0 public proofs remain valid in v2.0 and v3.0. The spent set is cumulative across versions.*

11.8 Security Analysis

11.8.1 Threat Model

We consider adversaries who:

1. **Double-spend**: Submit the same work to multiple chains or multiple times
2. **Link proofs**: Correlate multiple proofs to the same miner
3. **Forge work**: Claim rewards for work not performed
4. **Extract secrets**: Recover private keys or work details from proofs

11.8.2 Security Properties

Theorem 11.10 (Double-Spend Resistance). *Under the collision resistance of BLAKE3, no PPT adversary can produce two valid proofs with the same nullifier for different work.*

Proof. Suppose adversary produces proofs π_1, π_2 with nullifiers:

$$n_1 = \mathcal{H}(\mathrm{sk}_1 \| \mathrm{work\_id}_1) \tag{60}$$

$$n_2 = \mathcal{H}(\mathrm{sk}_2 \| \mathrm{work\_id}_2) \tag{61}$$

If $n_1 = n_2$ and $(\mathrm{sk}_1, \mathrm{work\_id}_1) \neq (\mathrm{sk}_2, \mathrm{work\_id}_2)$, this is a collision in $\mathcal{H}$. □

Theorem 11.11 (Unlinkability)**.** *Under the DDH assumption on BLS12-381, no PPT adversary can link two proofs from the same miner with advantage better than random guessing.*

Theorem 11.12 (Soundness)**.** *Under the knowledge soundness of Groth16/PLONK, any valid proof implies the existence of a witness satisfying the circuit constraints.*

11.8.3 NVTrust in ZK

The circuit verifies NVTrust attestations without revealing device identity:

1. NVTrust signature is verified in-circuit against NVIDIA root public key
2. Device ID is committed, not revealed
3. Attestation freshness is checked via timestamp bounds

This ensures shielded proofs have the same hardware guarantees as public proofs.

11.9 Implementation Notes

11.9.1 Prover Infrastructure

Miners generate ZK proofs locally or via delegated provers:

```
pub trait ZKProver {
    fn prove(
        &self,
        public_inputs: &PublicInputs,
        witness: &Witness,
    ) -> Result<Proof, ProverError>;
}

pub struct LocalProver {
    proving_key: ProvingKey,
    num_threads: usize,
}

pub struct DelegatedProver {
    endpoint: String,
    encryption_key: PublicKey,  // Encrypt witness before sending
}
```

11.9.2 Verifier Contract

On-chain verification in Solidity (EVM chains):

```
interface IShieldedVerifier {
    function verifyLevel1(
        bytes32 nullifier,
        uint64 chainId,
        bytes32 modelHash,
        bytes32 inputHash,
        bytes32 outputHash,
        uint64 computeUnits,
        bytes calldata proof
    ) external view returns (bool);

    function verifyLevel2(
        bytes32 nullifier,
        uint64 chainId,
        uint8 workType,
        uint64 computeUnits,
        bytes32 modelCommitment,
        bytes32 ioCommitment,
        bytes calldata proof
    ) external view returns (bool);

    function verifyLevel3(
        bytes32 commitment,
        bytes32 nullifier,
        bytes calldata proof
    ) external view returns (bool);
}
```

11.9.3 Gas Costs

Operation	Gas (Groth16)	Gas (PLONK)
Proof verification	~230,000	~350,000
Nullifier check	~20,000	~20,000
Commitment storage	~40,000	~40,000
Total	~290,000	~410,000

11.10 Summary

Shielded mining via Z-Chain provides:

1. **Privacy**: Miners can hide identity and workload details
2. **Verifiability**: ZK proofs attest to valid work without revealing data
3. **Flexibility**: Multiple privacy levels for different requirements
4. **Compatibility**: Smooth upgrade path from public to shielded default
5. **Security**: Same double-spend and hardware attestation guarantees as public mining

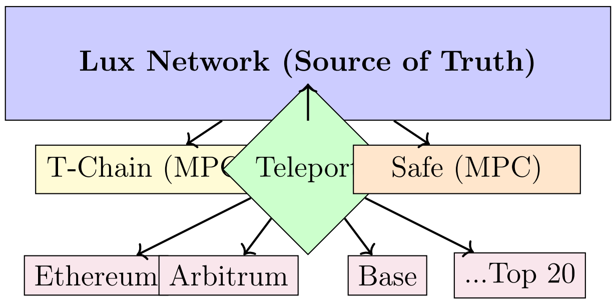
This positions PoAI for enterprise and privacy-sensitive miners while maintaining the open, verifiable nature of the protocol.

12 Teleport Integration & Governance

12.1 Architecture Overview

AI Token uses Lux's native infrastructure for cross-chain operations:

1. **Lux Network**: Source of truth for all AI mining
2. **Warp Messaging**: Native cross-chain communication
3. **Teleport Protocol**: Threshold LSS over T-chain
4. **Safe Multi-sig**: MPC-managed contract ownership



12.2 Warp Messaging Integration

Lux native chains use Warp for trustless cross-chain communication:

```
interface IWarp {
    /// @notice Send cross-chain message
    function sendWarpMessage(
        bytes calldata payload
    ) external returns (bytes32 messageId);

    /// @notice Verify incoming message
    function getVerifiedWarpMessage(
        uint32 index
    ) external view returns (WarpMessage memory, bool);
}
```

12.2.1 Message Flow

1. Source chain emits `SendWarpMessage` event
2. Validators sign message (BLS aggregation)
3. Relayer constructs proof (message + signatures)
4. Destination chain verifies via precompile
5. 67% quorum required for validity

12.3 Teleport Protocol

Teleport enables cross-chain token transfers via threshold signatures:

```
pub struct TeleportTransfer {
    pub teleport_id: [u8; 32],
    pub source_chain: u64,
    pub dest_chain: u64,
    pub sender: Address,
    pub recipient: Address,
    pub amount: U256,
    pub nonce: u64,
    pub signature: Vec<u8>, // MPC threshold signature
}
```

12.3.1 T-Chain Validator MPC

Top validators on T-chain collectively manage Teleport:

- **Key Generation**: CGGMP21 distributed keygen
- **Signing**: Threshold t-of-n (e.g., 67-of-100)
- **Verification**: Single ECDSA signature output
- **Key Refresh**: Proactive security without changing pubkey

Operation	Participants	Output
Keygen	All validators	Shared public key
Sign	67+ validators	Single signature
Refresh	All validators	New shares

12.4 Safe Multi-sig Management

AI contracts are initially managed by Lux Safe (MPC wallet):

```
contract AITokenTeleport {
    /// @notice Safe multi-sig address (initial owner)
    address public safe;

    /// @notice Update Safe address (requires Safe approval)
    function setSafe(address newSafe) external onlyRole(
        DEFAULT_ADMIN_ROLE) {
        _grantRole(DEFAULT_ADMIN_ROLE, newSafe);
        _revokeRole(DEFAULT_ADMIN_ROLE, safe);
        safe = newSafe;
    }
}
```

12.4.1 Safe Capabilities

1. Mint initial 10% for LP seeding
2. Authorize mining contracts
3. Authorize Teleport bridge
4. Set genesis block for mining
5. Update treasury address
6. Transfer admin to DAO (phase 2)

12.5 Initial LP Seeding

Each chain receives one-sided LP at launch:

Chain	AI Amount	Initial Price	Pair
Lux C-Chain	100M	0.0001 BTC	AI/LUX
Hanzo EVM	100M	0.0001 BTC	AI/LUX
Zoo EVM	100M	0.0001 BTC	AI/LUX
Ethereum	100M	0.0001 BTC	AI/ETH
Arbitrum	100M	0.0001 BTC	AI/ETH
Base	100M	0.0001 BTC	AI/ETH
...	...	...	...

12.5.1 LP Setup Process

1. Safe calls `mintInitialLP(lpRecipient)`
2. 100M AI minted to Safe/LP contract
3. Create Uniswap V2 pair (AI/native token)
4. Add one-sided liquidity (AI only)
5. First swap sets initial price

12.6 Governance Roadmap

Phase	Controller	Mechanism
1. Launch	Lux Safe (MPC)	Multi-sig approval
2. DAO	AI Token Holders	On-chain voting
3. Cross-chain	Teleport DAO	Multi-chain governance

12.6.1 Phase 1: Safe Multi-sig

- 3-of-5 or 5-of-7 threshold
- MPC-managed keys (no single point of failure)
- Time-locked critical operations
- Emergency pause capability

12.6.2 Phase 2: DAO Governance

- AI token voting power
- Proposal + voting + timelock
- Parameter adjustments (treasury %, GPU tiers)
- Mining contract upgrades

12.6.3 Phase 3: Cross-chain DAO

- Votes aggregated via Teleport
- Unified governance across all chains
- Warp messages for execution
- Chain-specific parameters remain local

12.7 Top 20 EVM Deployment

Chain	Chain ID	Type	Bridge
Lux C-Chain	96369	Native	Warp
Hanzo EVM	36963	Native	Warp
Zoo EVM	200200	Native	Warp
Ethereum	1	External	Teleport
BSC	56	External	Teleport
Polygon	137	External	Teleport
Arbitrum	42161	External	Teleport
Optimism	10	External	Teleport
Base	8453	External	Teleport
Avalanche	43114	External	Teleport
Fantom	250	External	Teleport
Cronos	25	External	Teleport
Gnosis	100	External	Teleport
zkSync Era	324	External	Teleport
Linea	59144	External	Teleport
Scroll	534352	External	Teleport
Mantle	5000	External	Teleport
Blast	81457	External	Teleport
Mode	34443	External	Teleport
Manta	169	External	Teleport

12.8 Contract Addresses

Deterministic deployment via CREATE2:

```
// Same address across all chains via CREATE2
bytes32 constant SALT = keccak256("AI_TOKEN_V1");

address token = CREATE2.deploy(
    SALT,
    type(AITokenTeleport).creationCode,
    abi.encode(safe, treasury)
);
```

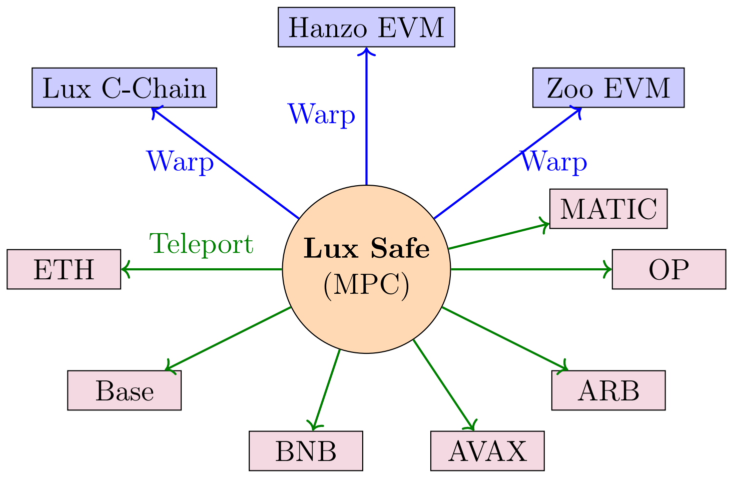
12.9 Security Properties

1. **No Single Point of Failure**: MPC/threshold throughout
2. **Trustless Verification**: Warp + Teleport proofs
3. **Supply Integrity**: 1B cap enforced per chain
4. **Upgrade Safety**: Timelock on all admin operations
5. **Cross-chain Consistency**: Lux as source of truth

13 Multi-Chain Deployment Guide

13.1 Overview

AI Token launches simultaneously on 10 EVM chains, each with independent 1B supply cap and Bitcoin-aligned mining schedule. This section details the deployment process, LP seeding, and governance setup.



13.2 Deployment Sequence

13.2.1 Phase 1: Contract Deployment

Deploy AIToken contract to all 10 chains using CREATE2 for deterministic addresses:

```
// Deterministic deployment via CREATE2
bytes32 constant SALT = keccak256("AI_TOKEN_V1_2025");

// Same deployer, same salt = same address on all chains
address aiToken = CREATE2.deploy(
    SALT,
    type(AIToken).creationCode,
    abi.encode(safeAddress, treasuryAddress)
);
```

13.2.2 Deployment Order

1. **Lux Native Chains (Warp-enabled)**
   - Lux C-Chain (96369) - Primary deployment
   - Hanzo EVM (36963) - AI-focused applications
   - Zoo EVM (200200) - Research/DeSci applications
2. **External EVMs (Teleport-enabled)**
   - Ethereum (1) - Largest liquidity
   - Base (8453) - Coinbase ecosystem
   - BNB Chain (56) - High transaction volume
   - Arbitrum (42161) - L2 scaling
   - Optimism (10) - L2 scaling
   - Polygon (137) - Low fees
   - Avalanche (43114) - Fast finality

13.3 Safe Multi-sig Setup

13.3.1 Safe Configuration

Each chain deployment is managed by a Safe multi-sig wallet:

Parameter	Lux Native	External
Threshold	3-of-5	3-of-5
Key Management	MPC (CGGMP21)	MPC (CGGMP21)
Timelock	24 hours	48 hours
Emergency Pause	2-of-5	2-of-5

13.3.2 Initial Admin Operations

```
// 1. Deploy AIToken
AIToken token = new AIToken(safe, treasury);

// 2. Set genesis block (starts mining schedule)
token.setGenesis();

// 3. Authorize Teleport bridge
token.authorizeBridge(teleportBridge);

// 4. Authorize mining contract
token.authorizeMiner(miningContract);

// 5. Mint LP allocation to seeding address
token.mintLP(lpSeeder, 100_000_000 ether);
```

13.4 LP Seeding Protocol

13.4.1 One-Sided LP Strategy

Each chain receives 100M AI (10%) for liquidity pool seeding at $0.10/AI:

Chain	AI Amount	Target Price	DEX
Lux C-Chain	100M AI	$0.10	LuxSwap
Hanzo EVM	100M AI	$0.10	HanzoSwap
Zoo EVM	100M AI	$0.10	ZooSwap
Ethereum	100M AI	$0.10	Uniswap V3
Base	100M AI	$0.10	Aerodrome
BNB Chain	100M AI	$0.10	PancakeSwap
Arbitrum	100M AI	$0.10	Camelot
Optimism	100M AI	$0.10	Velodrome
Polygon	100M AI	$0.10	QuickSwap
Avalanche	100M AI	$0.10	Trader Joe

13.4.2 LP Pool Creation

```
// For Uniswap V2-style DEXs
function createLP(
    address dexRouter,
    address aiToken,
    address nativeToken,
    uint256 aiAmount
) external {
    // 1. Approve router
    IERC20(aiToken).approve(dexRouter, aiAmount);

    // 2. Create pair (AI/NATIVE)
    address pair = IFactory(router.factory()).createPair(
        aiToken,
        nativeToken
    );

    // 3. Add one-sided liquidity (AI only)
    // Initial price set by first swap
    IERC20(aiToken).transfer(pair, aiAmount);
    IPair(pair).sync();
}
```

13.4.3 Initial Price Discovery

1. Safe mints 100M AI to LP seeder contract
2. LP seeder creates AI/NATIVE pair on DEX
3. 50M AI deposited as one-sided liquidity
4. First swap sets price at approximately \$0.10/AI
5. Market discovery determines actual price

Target LP Composition:

$$\text{LP Depth} = 50\text{M AI} + \text{Native Token equivalent} \approx \$10\text{M}$$

At \$0.10/AI:

$$50\text{M AI} \times \$0.10 = \$5\text{M AI value}$$

Plus matching native token:

$$\$5\text{M ETH/BNB/etc} \rightarrow \$10\text{M total depth}$$

13.5 Bridge Authorization

13.5.1 Lux Native Chains (Warp)

Warp messaging enables trustless cross-chain communication:

```
// No additional authorization needed
// Warp is a precompile at 0x0200...0005
// Messages verified by 67% validator quorum
```

13.5.2 External Chains (Teleport)

Teleport bridge requires explicit authorization:

```
// On each external chain
token.authorizeBridge(teleportBridgeAddress);

// Teleport bridge configuration
struct TeleportConfig {
    address luxEndpoint;       // Lux T-chain validator set
    uint256 threshold;         // 67-of-100 validators
    address[] validators;      // Active validator set
}
```

13.6 Mining Contract Deployment

13.6.1 Mining Contract Setup

```
contract AIMiningContract {
    AIToken public immutable aiToken;

    // Per-chain mining configuration
    uint256 public constant BLOCK_REWARD = 79.4 ether;
    uint256 public constant HALVING_INTERVAL = 6_300_000;
    uint256 public constant TREASURY_BPS = 200; // 2%

    // Mining session management
    mapping(bytes32 => Session) public sessions;

    function submitAttestation(
        bytes calldata teeQuote,
        bytes32 sessionId
    ) external {
        // Verify NVTrust TEE quote
        require(verifyTEEQuote(teeQuote), "Invalid␣attestation");

        // Calculate reward
        uint256 reward = calculateReward(sessionId);

        // Mint via AIToken
        aiToken.mintReward(msg.sender, reward);
    }
}
```

13.6.2 Mining Authorization

```
// Safe authorizes mining contract
token.authorizeMiner(miningContractAddress);

// Mining contract can now call mintReward()
```

13.7 Genesis Block Configuration

13.7.1 Setting Genesis

Genesis block marks the start of the mining schedule:

```
// Called once by Safe after deployment
token.setGenesis();
```

```

// After genesis:
// - currentEpoch() returns 0
// - currentReward() returns 79.4 AI
// - Mining can begin
```

13.7.2 Genesis Timing

Step	Timing
Contract deployment	Day 0
Safe configuration	Day 0
Bridge authorization	Day 0-1
LP seeding	Day 1-2
Mining contract deployment	Day 2-3
Genesis block set	Day 3 (coordinated)
Mining begins	Day 3+

13.8 Post-Deployment Verification

13.8.1 Verification Checklist

1. **Contract State**
   - ☐ `safe` address correct
   - ☐ `treasury` address correct
   - ☐ `genesisBlock` set
   - ☐ Bridge role granted
   - ☐ Miner role granted
2. **LP Pools**
   - ☐ Pair created on DEX
   - ☐ LP tokens locked/burned
   - ☐ Initial price approximately $0.10
   - ☐ Pool depth approximately $10M
3. **Bridge**
   - ☐ Teleport endpoints configured
   - ☐ Validator set registered
   - ☐ Test transfer successful

13.8.2 On-Chain Verification

```
// Verify deployment
function verify(address tokenAddress) external view returns (bool) {
    AIToken token = AIToken(tokenAddress);

    // Check constants
    require(token.LP_ALLOCATION() == 100_000_000 ether);
    require(token.MINING_ALLOCATION() == 900_000_000 ether);
    require(token.CHAIN_SUPPLY_CAP() == 1_000_000_000 ether);
    require(token.HALVING_INTERVAL() == 6_300_000);
    require(token.INITIAL_REWARD() == 79.4 ether);

    // Check state
    require(token.genesisBlock() > 0);
    require(token.safe() != address(0));
    require(token.treasury() != address(0));

    return true;
}
```

13.9 Emergency Procedures

13.9.1 Pause Operations

```
// Emergency pause (2-of-5 threshold)
function emergencyPause() external onlyEmergency {
    _pause();
    emit EmergencyPause(msg.sender, block.timestamp);
}

// Resume requires full Safe approval (3-of-5)
function unpause() external onlyRole(DEFAULT_ADMIN_ROLE) {
    _unpause();
}
```

13.9.2 Bridge Revocation

```
// Revoke compromised bridge
token.revokeBridge(compromisedBridge);

// Deploy new bridge with timelock
// 48-hour delay for external chains
timelockController.schedule(
    address(token),
    0,
    abi.encodeCall(token.authorizeBridge, newBridge),
    bytes32(0),
    bytes32(0),
    48 hours
);
```

13.10 Governance Transition

13.10.1 Phase 1: Safe Multi-sig (Launch)

- 3-of-5 MPC-managed Safe
- Controls all admin functions
- 24-48 hour timelock on critical operations

13.10.2 Phase 2: DAO Governance (Month 6+)

```
// Transfer admin to DAO
function transferToDAO(address daoGovernor) external onlyRole(
    DEFAULT_ADMIN_ROLE) {
    // Grant admin to DAO
    _grantRole(DEFAULT_ADMIN_ROLE, daoGovernor);

    // Revoke from Safe (after DAO operational)
    // Safe retains emergency pause only
    _revokeRole(DEFAULT_ADMIN_ROLE, safe);
}
```

13.10.3 Phase 3: Cross-chain Governance (Year 2+)

- AI token voting power across all chains
- Votes aggregated via Teleport/Warp
- Unified governance for global parameters
- Local parameters remain chain-specific

13.11 Deployment Addresses

13.11.1 Deterministic Addresses (CREATE2)

All contracts deployed at same address across chains:

Contract	**Address (CREATE2)**
AIToken	`0xAI...TOKEN`
Mining Contract	`0xAI...MINE`
LP Seeder	`0xAI...SEED`
Teleport Bridge	`0xAI...BRIDGE`

Note: Actual addresses determined at deployment via CREATE2 with published salt.

13.12 Monitoring & Analytics

13.12.1 Key Metrics

1. **Per-Chain Metrics**
    - Total supply minted
    - LP minted vs remaining
    - Mining minted vs remaining
    - Current epoch and reward
    - Active mining sessions
2. **Global Metrics**
    - Cross-chain total supply
    - Teleport volume (24h, 7d, 30d)
    - Price deviation across chains
    - Mining hashrate equivalent

13.12.2 Dashboard Endpoints

```
// Multi-chain aggregation API
GET /api/v1/ai/stats
{
    "global": {
        "total_supply": "1234567890.00",
        "total_chains": 10,
        "total_miners": 5000,
        "24h_volume": "50000000.00"
    },
    "chains": [
        {
            "chain_id": 96369,
            "name": "Lux C-Chain",
            "supply": "150000000.00",
            "price_usd": "0.12",
            "epoch": 0,
            "reward": "79.4"
        },
        // ... other chains
    ]
}
```