Article

Not peer-reviewed version

ExecMesh: A Compute-Backed Financial Infrastructure for the AI Economy

Panagiotis Karmiris *

Posted Date: 14 November 2025

doi: 10.20944/preprints202511.1085.v1

Keywords: zero-knowledge proofs; decentralized finance; ERC-1155; verifiable compute; recursive SNARKs; collateralization; L2 scalability; AI agent economy

 doi:10.20944/preprints202511.1085.v1

Article

ExecMesh: A Compute-Backed Financial Infrastructure for the AI Economy

Panagiotis Karmiris

Independent Researcher, Greece; unbinder@msn.com

Abstract

ExecMesh introduces two fundamental innovations to blockchain infrastructure: *compute credits as DeFi primitives* and *recursive proof composition for verifiable multi-agent workflows*. By transforming verified computational output into tradeable, collateralizable assets and enabling cryptographic composition of work across multiple agents, ExecMesh creates the foundational layer for a compute-backed economy. Global compute markets exceed $500B annually; ExecMesh makes this value programmable.

Keywords: zero-knowledge proofs; decentralized finance; ERC-1155; verifiable compute; recursive SNARKs; collateralization; L2 scalability; AI agent economy

Key Contribution: A secure collateralization system enabling compute credits as loan collateral with multi-source oracle protection. A compute futures market for hedging and speculation on computational capacity. Recursive proof composition enabling verifiable multi-step AI workflows. An economic model where computational output backs financial instruments. Graduated liquidation safeguards and post-quantum migration roadmap.

---

1. Introduction

1.1. The Problem

- **Verification problem:** When AI agents perform work (data labeling, model inference, web scraping), requesters have no cryptographic guarantee the work was actually done. This creates a trust bottleneck that prevents large-scale coordination.
- **Capital inefficiency:** Agents with proven computational capability cannot monetize this capability until they find work. Their demonstrated capacity sits idle, generating no return. Requesters who need compute in the future have no mechanism to secure capacity in advance.
- **Composition problem:** Complex AI tasks require multiple specialized agents working in sequence (scrape → analyze → report → visualize). Current systems cannot cryptographically prove that each step correctly used outputs from previous steps, making multi-agent workflows unverifiable.

1.2. The Opportunity

Three technological shifts create a new opening:

1. **AI proliferation** – exponential agent growth drives demand for verifiable work coordination.
2. **Layer-2 scalability** – sub-$0.01 transactions make on-chain verification economical.
3. **DeFi maturity** – existing lending protocols can accept new collateral types with proper risk management.

ExecMesh transforms verified compute into programmable collateral and enables the formation of *verifiable computational supply chains*.

 doi:10.20944/preprints202511.1085.v1

2. Design Goals and Assumptions

2.1. Design Goals

G1: Verifiable Execution Without Trust. Each task's correctness must be provable via succinct cryptographic evidence without reliance on intermediaries.

G2: Capital Efficiency for Agents. Agents with verified capacity can use compute credits as collateral to obtain capital with graduated risk management.

G3: Composable Workflows. Complex pipelines involving multiple agents are verifiable end-to-end through proof composition.

G4: Economic Sustainability. Fees and incentives ensure self-sustaining operations with realistic bootstrap strategy.

G5: Security First. Multi-source oracles, graduated liquidation, and post-quantum migration roadmap.

2.2. Assumptions

Assumption 1 (Honest Majority)**.** *Most agents act rationally to preserve reputation; slashing and challenge periods deter malicious behavior.*

Assumption 2 (Oracle Reliability)**.** *Price feeds from multiple independent sources with maximum 15% deviation per update; fallback to last known good price with circuit breaker.*

Assumption 3 (L2 Viability)**.** *Transaction costs on rollups remain below $0.10 per proof verification.*

Assumption 4 (zkSNARK Security)**.** *Groth16 on BN254 remains secure under the discrete log assumption; at least one trusted setup participant is honest.*

3. System Architecture

3.1. Overview

The ExecMesh protocol consists of five interconnected layers that together realize the *Proof-of-Execution* (PoX) system:

1. **Core Protocol Layer** – intent registry, escrow, reputation staking.
2. **Verification Layer** – zkSNARK validation and recursive composition.
3. **Financial Primitive Layer** – collateral vault, compute futures, AMM liquidity with secure oracles.
4. **Oracle Layer** – decentralized multi-source pricing aggregation.
5. **Application Layer** – web wallet, CLI, SDK interfaces.

Listing 1: ExecMesh Layered Architecture

```
+-------------------------------------------------------------+
|                        APPLICATION LAYER                    |
|  Web Wallet • CLI • SDK • Analytics Dashboard               |
+--------------------+----------------------+-----------------+
                     |                      |
+--------------------v----------------------+-----------------+
|                   FINANCIAL PRIMITIVE LAYER                 |
|   Collateral Vault • Futures Market • Secure Oracle         |
+--------------------+----------------------+-----------------+
                     |                      |
+--------------------v----------------------+-----------------+
|                       CORE PROTOCOL LAYER                   |
|   ProofOfWorkChannel • Reputation • Compute Credits         |
+--------------------+----------------------+-----------------+
                     |                      |
+--------------------v----------------------+-----------------+
|                       VERIFICATION LAYER                    |
|   Simple Proof (Groth16) • Composed Proof • Challenges      |
```

 doi:10.20944/preprints202511.1085.v1


```
+--------------------+----------------------+-------------------+
                     |                      |
+--------------------v----------------------+-------------------+
|                       INFRASTRUCTURE LAYER                     |
|   Ethereum L2 (Optimism) • Intent Matcher • Multi-Source Data|
+----------------------------------------------------------------+
```

3.2. Core Protocol Layer

The core contract, `ProofOfWorkChannel.sol`, manages job postings and escrow. Participants interact through an *intent lifecycle*:

1. **Post Intent:** Requester specifies reward, deadline, and capability.
2. **Accept:** Agent stakes reputation and agrees to perform work.
3. **Complete:** Agent executes the job off-chain and generates a zk proof.
4. **Submit Proof:** Proof and output hash are recorded on-chain.
5. **Challenge Period:** A 1-hour window permits disputes.
6. **Settle:** Escrow releases payment; compute credits are minted.

Reputation and compute credits both use ERC-1155 token standards. Reputation tokens are non-transferable and subject to slashing; compute credits are transferable receipts representing verified capacity.

3.3. Verification Layer

3.3.1. Simple Proof Verification

For atomic tasks, the protocol validates Groth16 proofs [1]. Given circuit C, inputs (x, w), and verification key vk:

$$\mathsf{Verify}(\mathsf{vk}, x, \pi) = 1 \iff \exists w : C(x, w) = 1$$

On success, the transaction finalizes and credits are minted.

3.3.2. Composed Proof Verification

Complex workflows are chains of dependent tasks. Let π_i denote the proof for step i with public hash $h_i = H(o_i)$, where o_i is the step's output. A *composition proof* π_c demonstrates that the prover knew the actual outputs $\{o_i\}$ and correctly computed a new result o_c.

Definition 1 (Composed Proof)**.** *Given dependency hashes $\{h_i\}$, a composed proof π_c is valid if:*

$$H(o_i) = h_i \; \forall i, \quad H(f(\{o_i\})) = h_c,$$

and π_c proves knowledge of $\{o_i\}$ and transformation f satisfying the above.

Verification Logic (Solidity).

Listing 2: Verification Logic (Solidity)

```
for each dependency in deps:
    require(isSettled(dependency.intentId));
    bytes32 hash = proofs[dependency.intentId].outputHash;
    require(verifyInclusion(hash, dependency.inputHash));
require(zkVerify(composeProof));
```

3.3.3. ZK Circuit for Composition

Implemented in Circom:

 doi:10.20944/preprints202511.1085.v1

Listing 3: composition.circom (simplified)

```
template CompositionVerifier(N) {
    signal input outputHash;
    signal input depHashes[N];
    signal input deps[N];      // private inputs
    signal input output;       // private
    signal input weights[N];   // transformation vector

    // Check dependency hashes
    component hashers[N];
    for (var i=0; i<N; i++) {
        hashers[i] = Poseidon(1);
        hashers[i].inputs[0] <== deps[i];
        hashers[i].out === depHashes[i];
    }

    // Apply transformation
    signal computed;
    computed <== 0;
    for (var j=0; j<N; j++) {
        computed <== computed + deps[j]*weights[j];
    }

    // Verify output hash
    component outH = Poseidon(1);
    outH.inputs[0] <== computed;
    outH.out === outputHash;
}
```

What it Proves.

The prover knows dependency data that hash to the declared values and that the final output was derived by an agreed computation f. Actual data remain private; only hashes and succinct proof are public.

3.4. Gas Optimization Strategies

- **Batch verification:** aggregate multiple proofs per tx.
- **Proof recursion:** nested Groth16 (phase 2).
- **Calldata compression:** packed bytes for proof elements.
- **L2 deployment:** Optimism/Arbitrum yields 100× cost reduction.

Scenario	Gas	Cost (L2 est.)
Simple proof	150k	$0.005
3-dependency composition	280k	$0.009
Batch of 10	800k	$0.027

4. Financial Primitives

ExecMesh transforms verified computation into programmable capital through a set of smart-contract primitives deployed on Layer 2 networks. These extend traditional DeFi mechanisms by introducing *compute-backed assets*.

4.1. Compute Credit Vault

The `ComputeCreditVault` contract locks ERC-1155 compute credits as collateral and interfaces with external lending protocols such as Aave or Compound.

Data Structure.

 doi:10.20944/preprints202511.1085.v1

Listing 4: Locked position structure (production implementation)

```
struct LockedPosition {
    uint256 amount;         // Number of credits locked
    uint256 debt;           // Outstanding debt amount
    address protocol;       // DeFi protocol (Aave, Compound)
    uint256 lockTime;       // When position was created
    uint256 warningTime;    // Grace period timestamp
    bytes32 purposeHash;    // Purpose or loan identifier
    bool active;            // Position status
}
```

Lifecycle.

1. Agent mints compute credits after verified completion.
2. Credits are deposited into the Vault via `lockAsCollateral`.
3. Lending protocol queries value through secure adapter contract.
4. Borrower obtains liquidity up to an LTV ratio r (typically 66%).
5. Health factor monitored continuously with graduated liquidation.
6. Upon repayment, credits are unlocked and returned.

Adapter Interface.

Listing 5: Aave Compute Adapter (corrected)

```
contract AaveComputeAdapter {
    ComputeCreditVault public vault;
    ComputeCreditOracle public oracle;
    uint256 constant LTV_RATIO = 66;

    function getMaxBorrow(address user, uint256 tokenId)
        external view returns (uint256)
    {
        uint256 v = vault.getCollateralValue(user, tokenId);
        return (v * LTV_RATIO) / 100;
    }

    function checkHealth(address user, uint256 tokenId)
        external view returns (bool)
    {
        uint256 health = vault.getHealthFactor(user, tokenId);
        return health >= 1.0e18; // 100% collateralization minimum
    }
}
```

4.2. *Compute Futures Market*

A secondary contract, `ComputeFutures`, allows trading of future compute commitments.

Listing 6: Compute future struct

```
struct ComputeFuture {
    address seller;             // Agent committing to deliver compute
    CapabilityType capability;  // GPU, Storage, API, etc.
    uint256 computeUnits;       // Promised capacity
    uint256 pricePerUnit;       // Locked price
    uint256 expiryTime;         // Deadline
    address buyer;              // Purchaser (optional)
    bool settled;               // Delivery confirmed
    uint256 creationTime;       // For grace period calculations
}
```

 doi:10.20944/preprints202511.1085.v1

Use Cases.

- **Hedging.** Requesters fix future compute prices.
- **Speculation.** Traders arbitrage between spot and future prices.
- **Yield.** Agents monetize idle capacity ahead of delivery.

Settlement occurs when a verified intent corresponding to the future is submitted; failure to deliver results in forfeiture and reputation slashing.

4.3. Secure Oracle System

Pricing of compute credits relies on a secure multi-source oracle service that aggregates data with robust safeguards:

Listing 7: Secure Multi-Source Oracle (production implementation)

```
contract ComputeCreditOracle {
    uint256 constant MAX_DEVIATION = 15; // 15% maximum price change
    uint256 constant MIN_SOURCES = 3;    // Minimum independent sources
    uint256 constant STALENESS_THRESHOLD = 1 hours;
    uint256 constant MAX_SOURCE_CONCENTRATION = 40; // Max 40% per source

    struct PriceSource {
        address provider;    // Chainlink, Pyth, Uniswap TWAP, etc.
        uint256 price;
        uint256 timestamp;
        uint256 confidence;  // 0-100% confidence score (weight)
    }

    mapping(CapabilityType => uint256) public currentPrice;
    mapping(CapabilityType => uint256) public lastUpdateTime;

    function updatePrice(CapabilityType cap, PriceSource[] calldata sources)
        external onlyRole(ORACLE_UPDATER) whenNotPaused nonReentrant
    {
        require(sources.length >= MIN_SOURCES, "Insufficient sources");

        // Validate sources aren't stale
        for (uint i = 0; i < sources.length; i++) {
            require(
                block.timestamp - sources[i].timestamp < STALENESS_THRESHOLD,
                "Stale price data"
            );
        }

        // Calculate weighted median
        uint256 newPrice = _calculateWeightedMedian(sources);
        uint256 oldPrice = currentPrice[cap];

        // Check deviation
        if (oldPrice > 0) {
            uint256 deviation = _calculateDeviation(oldPrice, newPrice);
            require(deviation <= MAX_DEVIATION, "Price deviation too high");
        }

        // Check source diversity (no single source > 40%)
        require(_checkSourceDiversity(sources), "Source concentration risk");

        // Update with TWAP for stability
        currentPrice[cap] = _updateTWAP(cap, newPrice);
        lastUpdateTime[cap] = block.timestamp;

        emit PriceUpdated(cap, newPrice, oldPrice, sources.length, block.timestamp);
    }

```

 doi:10.20944/preprints202511.1085.v1


```
    function _checkSourceDiversity(PriceSource[] calldata sources)
        internal pure returns (bool)
    {
        uint256 totalWeight = 0;
        for (uint i = 0; i < sources.length; i++) {
            totalWeight += sources[i].confidence;
        }

        // Ensure no single source exceeds MAX_SOURCE_CONCENTRATION
        for (uint i = 0; i < sources.length; i++) {
            uint256 concentration = (sources[i].confidence * 100) / totalWeight;
            if (concentration > MAX_SOURCE_CONCENTRATION) {
                return false;
            }
        }
        return true;
    }
}
```

Oracle Data Sources & Weights

The protocol does not enforce fixed weights on-chain. Instead, it ensures no single source exceeds 40% of total confidence weight. This provides flexibility while preventing manipulation.

Recommended weights for oracle operators:

1. Historical ExecMesh marketplace data (30% suggested weight)
2. External cloud markets (AWS, GCP, Azure - 25% suggested)
3. Decentralized compute (Akash, Render - 20% suggested)
4. Uniswap V3 on-chain liquidity pools (15% suggested)
5. Futures market time-weighted average (10% suggested)

These are guidelines; the on-chain enforcement is the 40% concentration limit per source, providing both flexibility and security.

5. Economic Model

5.1. Conservative Market Analysis

- Global compute market: $500B+ (Gartner 2024)
- AI/ML compute segment: $50B growing 40% YoY
- Decentralized compute: $500M (1% penetration)
- **Addressable market for ExecMesh:** AI agent workflows = $5B by 2028

5.2. Bootstrap Strategy: From Zero to Critical Mass

Phase 1: Seeded Liquidity (Months 0-3)

- Protocol treasury provides $100k in initial liquidity incentives
- 10x rewards for first 100 agents and 50 requesters
- Zero fees for first 1,000 jobs
- Target: 100 jobs/month @ $50 average (realistic micro-tasks)
- Focus: Data labeling partnerships with AI training companies

Phase 2: Organic Growth (Months 3-12)

- Reduced incentives: 3x rewards for strategic capabilities
- Tiered fee structure:
  - Micro jobs (<$10): 0.1% (loss leader)
  - Standard ($10-$100): 1.0%
  - Enterprise (>$100): 0.5% (volume discount)

 doi:10.20944/preprints202511.1085.v1

- Target: 1,000 jobs/month by month 12
- Expand to research automation and content generation

Phase 3: Self-Sustaining (Year 2+)

- Standard fees: 1% average across all job sizes
- Target: 5,000-10,000 jobs/month
- Total addressable market: $50B compute market × 2% capture = $1B potential

5.3. Conservative Revenue Projections

Break-even Analysis

Fixed costs: $\$200K$/yr (team + infrastructure + security audits)

Variable costs: $\$50K$/yr (oracle services, gas subsidies)

Break-even: Month 18 (Year 2, Q2)

Profitability: Year 3+ with $900K annual revenue

Table 1. Realistic Three-Year Revenue Projection.

Phase	Jobs/Month	Avg. Value	Fee Rate	Monthly Revenue	Annual
Bootstrap (Y1)	100 → 1,000	$50	0.5%	$1,500	$18,000
Growth (Y2)	1,000 → 5,000	$75	1.0%	$22,500	$270,000
Scale (Y3)	5,000 → 10,000	$100	1.0%	$75,000	$900,000

5.4. Value Accrual

Agents earn ETH for jobs, reputation tokens for trust, and compute credits for collateralization or sale. **Requesters** gain verifiable output and predictable pricing. The **protocol** captures fees from transactions and liquidity pools.

5.5. Tokenomics

If governance token `$EXEC` is introduced, supply and allocation are:

Category	Share	Vesting
Community Rewards	30%	10 yr emission
Team & Contributors	20%	4 yr vesting
Treasury	20%	–
Investors	15%	2 yr vesting
Liquidity Mining	10%	–
Grants & Partnerships	5%	–

Utility includes governance, staking, and boosted rewards.

5.6. Fee Structure

Action	Fee	Notes
Job Settlement	0.1-1.0%	Tiered by size
Future Creation	0.1%	of notional value
Future Settlement	0.5%	of notional value
Collateral Unlock	0.1%	on release
Liquidation	2-5%	graduated penalties

5.7. Market Dynamics

Supply Side: Low entry barriers for agents; reputation and specialization drive premiums.

 doi:10.20944/preprints202511.1085.v1

Demand Side: Pay-per-use model with transparent, verifiable outcomes.

Equilibrium: Price discovery via futures, quality via reputation, capacity planning via collateral.

6. Security Model & Risk Mitigation

6.1. Enhanced Threat Model

We categorize threats by adversary class with graduated mitigations:

Malicious Agent: attempts to submit forged proofs or double-spend reputation.

Malicious Requester: disputes valid work to avoid payment.

Oracle Manipulator: feeds distorted price data to profit from liquidation.

Market Manipulator: abuses futures to influence spot pricing.

Quantum Adversary: future threat to cryptographic foundations.

6.2. Smart-Contract Security

- Reentrancy protection and checked arithmetic (Solidity ≥ 0.8).
- Access control via OpenZeppelin roles with multi-sig for critical functions.
- Pausable emergency circuit breakers with time-lock governance.
- Proxy upgradeability with 7-day timelock for major changes.
- Comprehensive test coverage ($> 95\%$) and formal verification for critical circuits.

6.3. Enhanced Liquidation Mechanisms

Multi-tier Liquidation Thresholds

Rather than a single 150% collateralization ratio, we implement graduated margins:

Table 2. Graduated Liquidation Safeguards.

Health Factor	State	Action	Penalty
$H > 1.5$	Healthy	None	0%
$1.3 < H \leq 1.5$	Warning	Email/on-chain alert	0%
$1.1 < H \leq 1.3$	At Risk	Collateral top-up required	0%
$1.0 < H \leq 1.1$	Critical	Partial liquidation (50%)	2%
$H \leq 1.0$	Underwater	Full liquidation	5%

Liquidation Safeguards Implementation

Listing 8: Safe Graduated Liquidation (production implementation)

```
function liquidate(address user, uint256 tokenId)
    external nonReentrant onlyRole(LIQUIDATOR_ROLE)
{
    LockedPosition storage position = positions[user][tokenId];
    require(position.active, "No active position");

    uint256 health = getHealthFactor(user, tokenId);
    require(health < AT_RISK_THRESHOLD, "Position is safe");

    // Grace period for moderate risk positions
    if (health >= CRITICAL_THRESHOLD && health < AT_RISK_THRESHOLD) {
        if (position.warningTime == 0) {
            position.warningTime = block.timestamp;
            emit WarningIssued(user, tokenId, health);
            return;
        }

        require(
```

 doi:10.20944/preprints202511.1085.v1


```
            block.timestamp >= position.warningTime + GRACE_PERIOD,
            "Grace period active"
        );
    }

    // Execute appropriate liquidation
    if (health >= CRITICAL_THRESHOLD) {
        _executePartialLiquidation(user, tokenId);
    } else {
        _executeFullLiquidation(user, tokenId);
    }
}
```

6.4. Oracle and Market Security

- Multi-source aggregation with weighted median calculation
- Maximum 15% deviation per update with circuit breaker
- 1-hour staleness threshold for price data
- Maximum 40% concentration per source to prevent manipulation
- Emergency freeze function for extreme market conditions

7. Competitive Positioning & Market Differentiation

7.1. Unique Value Proposition Matrix

Table 3. ExecMesh vs. Alternative Solutions.

Capability	ExecMesh	Akash	Render	Giza	Chainlink
Verifiable Compute	✓✓	✗	✗	✓	✗
Financial Primitives	✓✓	✗	✗	✗	Partial
Multi-Agent Workflows	✓✓	Partial	✗	✗	✗
Compute Futures	✓✓	✗	✗	✗	✗
EVM Native	✓✓	✗	✗	✗	✓✓
Proving Cost	$0.005	N/A	N/A	$0.10+	N/A
Time to Proof	<2s	N/A	N/A	10s+	N/A
Collateralization	✓✓	✗	✗	✗	✗

7.2. Strategic Focus: AI Agent Economy

While competitors focus on infrastructure replacement, ExecMesh targets the **emerging AI agent coordination market**:

Initial Wedge: Data Labeling Pipelines ($2B Market)

1. **Scrape:** Web data collection with proof of source integrity
2. **Label:** Human/AI labeling with quality verification proofs
3. **Verify:** Cross-validation with consensus proofs
4. **Train:** Model training with reproducible dataset proofs

Expansion Verticals

- **Research Automation:** Literature review → Analysis → Drafting → Peer review
- **Content Generation:** Research → Outline → Draft → Edit → Publish
- **Financial Analysis:** Data gathering → Modeling → Risk assessment → Reporting

7.3. Competitive Advantages

1. **First-mover in compute financialization** - No other protocol offers compute credits as DeFi collateral

 doi:10.20944/preprints202511.1085.v1

2. **Recursive proof composition** - Unique capability for multi-agent workflows
3. **EVM-native deployment** - Immediate compatibility with existing DeFi ecosystem
4. **Graduated risk management** - Superior collateral protection vs. traditional DeFi

8. Technical Implementation

8.1. Smart-Contract Suite

Target coverage $> 95\%$, gas < 300k per composed proof, formal verification for critical financial functions.

Listing 9: Enhanced Smart Contract Suite

```
ProofOfWorkChannel.sol     // v1 - deployed
   Intent management
   Reputation staking
   Optimistic settlement

ComputeCreditVault.sol     // v2 - enhanced
   Collateralization with health monitoring
   Graduated liquidation
   Insurance fund

ComputeCreditOracle.sol    // v2 - new secure oracle
   Multi-source aggregation
   Deviation limits & circuit breakers
   Emergency freeze

CompositionVerifier.sol    // v2 - new
   Recursive proof checks
   Dependency tracking
   Batched verification

ComputeFutures.sol         // v2 - new
   Future market creation
   Settlement with reputation
   Price discovery
```

8.2. Off-Chain Infrastructure

Intent Matcher: REST API + WebSocket for job discovery and ML-based agent scoring.

Proof Generator: Circom circuit compilation, witness and Groth16 proof creation with optimization.

Oracle Service: Multi-source price aggregation, anomaly detection, and secure on-chain updates.

Analytics Service: Market statistics, gas monitoring, health factor alerts.

Security Monitoring: 24/7 threat detection and emergency response system.

8.3. Frontend Applications

Web Wallet (React + TypeScript) for human interaction; CLI tool for agent automation; analytics dashboard built on Grafana + PostgreSQL with real-time health monitoring.

9. Roadmap & Future Work

Phase 1 (Completed): PoX v1, Merkle proofs, reputation system.

Phase 2 (Q4 2025): Deploy Secure Vault, Futures Market, Multi-source Oracle, Aave adapter.

Phase 3 (Q1 2026): Deploy composition circuits, launch workflow marketplace, data labeling partnerships.

Phase 4 (Q2–Q3 2026): Mainnet deployment (Optimism/Base), compute provider partnerships, formal verification.

 doi:10.20944/preprints202511.1085.v1

Phase 5 (Q4 2026+): Launch governance token, transition to DAO, expand to research automation.

9.1. Post-Quantum Cryptography Migration

Current Vulnerability

BN254 curve used in Groth16 is vulnerable to Shor's algorithm. Estimated break time: 1 hour on a 10M-qubit quantum computer (expected 2035+).

Migration Timeline

Table 4. Post-Quantum Migration Strategy.

Timeline	Technology	Action
2025-2026	Groth16 (BN254)	Current implementation
2027-2028	PLONK + BN254	Add alternative verifier
2029-2030	Lattice-based SNARKs	Pilot with new curves
2031+	STARKs/other PQ	Full migration

Contingency Plan

- Monitor NIST post-quantum standardization process
- Maintain contract upgradeability via proxy pattern
- Community governance vote on migration timing
- **Emergency freeze:** Protocol can be paused if quantum threat becomes imminent
- Research collaboration with academic institutions

10. Conclusions

ExecMesh v2.1 represents a fundamental shift in how we value computational work. By transforming verified compute into a financial primitive with robust security safeguards and enabling recursive proof composition, we create the infrastructure for a truly trustless AI economy. Our enhanced security model with multi-source oracles, graduated liquidation, and post-quantum migration roadmap addresses critical vulnerabilities while maintaining capital efficiency.

The protocol's initial focus on data labeling pipelines provides a concrete wedge into the $2B AI training market, with clear expansion paths to research automation and content generation. Our conservative economic projections and realistic bootstrap strategy ensure sustainable growth.

This is the foundational protocol for the AI economy, where work is verifiable, composable, and financially productive—creating a world where computational output becomes as programmable and valuable as capital itself.

Acknowledgments: This work builds upon research and implementations from Satoshi Nakamoto [2], Vitalik Buterin et al. [3], Gavin Wood [4], Jens Groth [1], Ben-Sasson et al. [5], Bünz et al. [6], and the Aave [7] and Uniswap [8] teams. Special thanks to the critical reviewers whose feedback significantly strengthened this whitepaper.

Appendix A. Mathematical Proofs & Formal Verification

Appendix A.1. Collateral Safety Theorem

Theorem A1 (Enhanced Collateral Safety)**.** *For any locked compute credit with collateralization ratio* $R > 150\%$ *and graduated liquidation thresholds, the vault remains solvent under price drops up to* $(R - 100)/R$ *with partial liquidation restoring health before full insolvency.*

Proof. Let V_0 be the initial credit value and D the debt issued. By definition $V_0 = R \times D$ with $R > 1.5$. The graduated liquidation system triggers at $H = 1.1$ (110% collateralization):

$$\text{Partial liquidation threshold: } V_{\text{partial}} = 1.1 \times D$$

 doi:10.20944/preprints202511.1085.v1

$$\text{Safe decline to partial liquidation: } \frac{V_0 - V_{\text{partial}}}{V_0} = \frac{RD - 1.1D}{RD} = \frac{R - 1.1}{R}$$

For $R = 1.5$, safe decline to partial liquidation = $\frac{1.5-1.1}{1.5} = 26.7\%$.

Partial liquidation then restores health to $H = 1.3$, providing additional buffer. The system remains solvent through this graduated approach. □

Appendix A.2. Proof Composition Soundness

Theorem A2 (Soundness of Composed Proofs). *If individual proofs $\pi_1, \ldots, \pi_n$ are valid and the composition proof π_c references their output hashes using collision-resistant hash function H, then acceptance of π_c implies the prover used the genuine dependency outputs with negligible probability of forgery.*

Proof. Let $D = \{d_i\}$ be dependency outputs and $h_i = H(d_i)$ their hashes. Each π_i proves knowledge of d_i such that $H(d_i) = h_i$ with soundness error ϵ_{zk}.

The composed proof π_c proves knowledge of inputs whose hashes equal $\{h_i\}$ and transformation producing output O.

By the collision-resistance of H, the probability of finding $d_i' \neq d_i$ such that $H(d_i') = h_i$ is negligible ϵ_{cr}.

Thus the probability that π_c verifies but uses incorrect dependencies is bounded by:

$$\Pr[\text{forgery}] \leq \epsilon_{\text{zk}} + \epsilon_{\text{cr}} + \text{negl}(\lambda)$$

Which is negligible in security parameter λ. □

Appendix A.3. Oracle Manipulation Resistance

Theorem A3 (Oracle Manipulation Resistance). *Given $n \geq 3$ independent price sources with at least $2/3$ honest sources, the weighted median calculation resists manipulation requiring control of more than $n/2$ sources.*

Proof. Let S be the set of price sources with $|S| = n$, and $H \subset S$ be the honest sources with $|H| \geq \lceil 2n/3 \rceil$.

The weighted median calculation sorts sources by price and selects the price where cumulative weight reaches 50%.

To manipulate the median downward, an attacker must control enough sources to push the cumulative weight of lower prices past 50%. This requires controlling sources with total weight $> 50\%$.

But honest sources have total weight $\geq 2/3 \times$ total weight $> 50\%$, so the median must include at least one honest source price.

Thus manipulation requires controlling $> 50\%$ of total weight, which is impossible with $2/3$ honest assumption. □

Appendix B. Circuit Specifications & Benchmarks

Appendix B.1. Cryptographic Parameters

Table A1. Enhanced Cryptographic Parameters for ExecMesh v2.1

Parameter	Value	Security Level	Rationale
zkSNARK System	Groth16	128-bit	Industry standard, battle-tested
Elliptic Curve	BN254 (alt_bn128)	128-bit	Native to EVM, gas-efficient
Hash Function	Poseidon	128-bit	ZK-friendly, optimized for circuits
Proof Size	256 bytes	Constant	Independent of circuit size
Verification Gas	∼150k gas	∼$0.005 on L2	Acceptable for settlements
Trusted Setup	Powers of Tau	Phase 1: Perpetual	Community ceremony
Security Margin	2^{128} operations	128-bit	Sufficient until quantum era
Quantum Timeline	2035+	Migration roadmap	Section 9.1

 doi:10.20944/preprints202511.1085.v1

Appendix B.2. Circuit Complexity Benchmarks

Table A2. Enhanced Circuit Complexity Benchmarks

Circuit	Constraints	Proving Time	Verification Time	Memory
Simple (compute.circom)	1	∼200ms	∼5ms	<100MB
Composition (5 deps)	17	∼500ms	∼8ms	∼500MB
Composition (20 deps)	62	∼2s	∼12ms	∼2GB
Oracle Verification	25	∼300ms	∼6ms	∼200MB
Liquidation Check	8	∼150ms	∼4ms	∼100MB

Appendix B.3. Gas Cost Analysis

Table A3. Detailed Gas Cost Analysis (L2 Optimism Estimates)

Operation	Description	Gas Used	Cost @ 0.001gwei
Simple Proof Verify	Groth16 verification	150,000	$0.005
Composition Verify	3 dependencies	280,000	$0.009
Batch Verify (10)	Aggregated proofs	800,000	$0.027
Oracle Update	Multi-source median	120,000	$0.004
Liquidation Check	Health calculation	80,000	$0.003
Future Settlement	With reputation check	200,000	$0.007
Collateral Lock	ERC-1155 approval	100,000	$0.003

Appendix C. Formal Security Specifications

Appendix C.1. Oracle Security Parameters

Deviation Limits

Maximum price change per update: 15%
Minimum time between updates: 10 minutes
Staleness threshold: 1 hour
Minimum sources: 3 independent providers
Maximum concentration: 40% per source

Recommended Source Weights (Guidelines for Operators)

$$\begin{aligned} w_{\text{execmesh}} &= 0.30 \quad \text{(historical market data)} \\ w_{\text{cloud}} &= 0.25 \quad \text{(AWS, GCP, Azure spot prices)} \\ w_{\text{decentralized}} &= 0.20 \quad \text{(Akash, Render prices)} \\ w_{\text{uniswap}} &= 0.15 \quad \text{(on-chain liquidity)} \\ w_{\text{futures}} &= 0.10 \quad \text{(time-weighted average)} \end{aligned}$$

Note: These are suggested weights. The protocol enforces only the 40% concentration limit on-chain.

Appendix C.2. Liquidation Safety Proofs

Health Factor Calculation

The health factor H for a position with collateral value V and debt D is:

$$H = \frac{V \times 10^{18}}{D}$$

 doi:10.20944/preprints202511.1085.v1

where the result is in 18 decimal precision (e.g., 1.5×10^{18} represents 150% collateralization).

Grace Period Justification

The 6-hour grace period for positions with $1.1 \times 10^{18} < H \leq 1.3 \times 10^{18}$ provides:

- Time for price oracle updates to reflect market conditions
- Opportunity for position owners to add collateral
- Prevention of premature liquidations during temporary volatility
- Balance between protection and capital efficiency

Appendix C.3. Emergency Scenarios & Response

Oracle Failure

1. Detect insufficient sources (<3 available)
2. Freeze new loans and liquidations
3. Use last known good price for up to 24 hours
4. Emergency governance vote for resolution

Market Manipulation Detection

- Monitor for coordinated price source deviations
- Check futures market arbitrage opportunities
- Analyze trading patterns for wash trading
- Implement circuit breaker if manipulation detected

Quantum Emergency

1. Monitor quantum computing milestones
2. Freeze protocol if Shor's algorithm becomes practical
3. Execute post-quantum migration via governance
4. Use insurance fund for any losses during transition

Appendix D. API Reference & Integration Guide

Appendix D.1. ComputeCreditVault API

- `lockAsCollateral(uint256 tokenId, uint256 amount, address protocol, bytes32 purposeHash)`
- `unlockCollateral(uint256 tokenId)`
- `getHealthFactor(address user, uint256 tokenId) returns (uint256)`
- `recordDebt(address user, uint256 tokenId, uint256 debtAmount)`
- `recordRepayment(address user, uint256 tokenId, uint256 repayAmount)`
- `liquidate(address user, uint256 tokenId) // liquidator role only`
- `getMaxBorrow(address user, uint256 tokenId) returns (uint256)`

Appendix D.2. Secure Oracle API

- `updatePrice(CapabilityType cap, PriceSource[] calldata sources)`
- `getPrice(CapabilityType cap) returns (uint256)`
- `getPriceWithAge(CapabilityType cap) returns (uint256 price, uint256 age)`
- `isStale(CapabilityType cap) returns (bool)`
- `addTrustedSource(address source, string calldata name) // admin only`
- `pause() // admin only`

Appendix D.3. Intent Matcher REST API

- `GET /api/v1/intents/open?capability={type}&maxReward={amount}`
- `GET /api/v1/futures/market?capability={type}&timeframe={days}`
- `GET /api/v1/agents/{address}/reputation`

 doi:10.20944/preprints202511.1085.v1

- `POST /api/v1/agents/register`
- `POST /api/v1/intents/create`
- `GET /api/v1/oracle/health // system status`

Appendix D.4. Web3 Integration Examples

Checking Collateral Health

Listing 10: Monitoring Position Health (corrected)

```
// Real-time health monitoring
const monitorHealth = async (userAddress, tokenId) => {
    const vault = new ethers.Contract(vaultAddress, vaultABI, provider);
    const health = await vault.getHealthFactor(userAddress, tokenId);

    if (health < ethers.utils.parseUnits("1.3", 18)) {
        // Send alert - position at risk
        const healthFormatted = ethers.utils.formatUnits(health, 18);
        sendAlert(userAddress, `Health factor: ${healthFormatted}`);
    }

    return health;
};
```

Secure Price Updates

Listing 11: Oracle Price Update Script

```
// Secure multi-source price aggregation
const updatePrices = async () => {
    const sources = await Promise.all([
        getChainlinkPrice(capability),
        getCloudSpotPrice(capability),
        getUniswapTWAP(capability),
        getFuturesMarketPrice(capability)
    ]);

    const validSources = sources.filter(s =>
        s.confidence > 50 && Date.now() - s.timestamp < 3600000
    );

    if (validSources.length >= 3) {
        const oracle = new ethers.Contract(oracleAddress, oracleABI, signer);
        await oracle.updatePrice(capability, validSources);
    }
};
```

References

1. Groth, J. On the Size of Pairing-Based Non-Interactive Arguments. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2016, EUROCRYPT 2016, pp. 305–326. https://doi.org/10.1007/978-3-662-49896-5_11.
2. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008. Accessed: 2025-11-01.
3. Buterin, V.; Griffith, V. Casper the Friendly Finality Gadget. In Proceedings of the arXiv preprint arXiv:1710.09437, 2019.
4. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper* **2014**, *151*, 1–32.
5. Ben-Sasson, E.; Chiesa, A.; Tromer, E.; Virza, M. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In Proceedings of the 23rd USENIX Security Symposium, 2014, pp. 781–796.

 doi:10.20944/preprints202511.1085.v1

6. Bünz, B.; Fisch, B.; Szepieniec, A. Transparent SNARKs from DARK Compilers. *IACR Cryptology ePrint Archive* **2020**, *2019*, 1229.
7. Aave Companies. Aave Protocol V3: Technical Paper. *Aave Documentation* **2023**. Available at: https://github.com/aave/aave-v3-core.
8. Adams, H.; Zinsmeister, N.; Salem, M.; Keefer, R.; Robinson, D. Uniswap V3 Core. *Uniswap Whitepaper* **2021**.