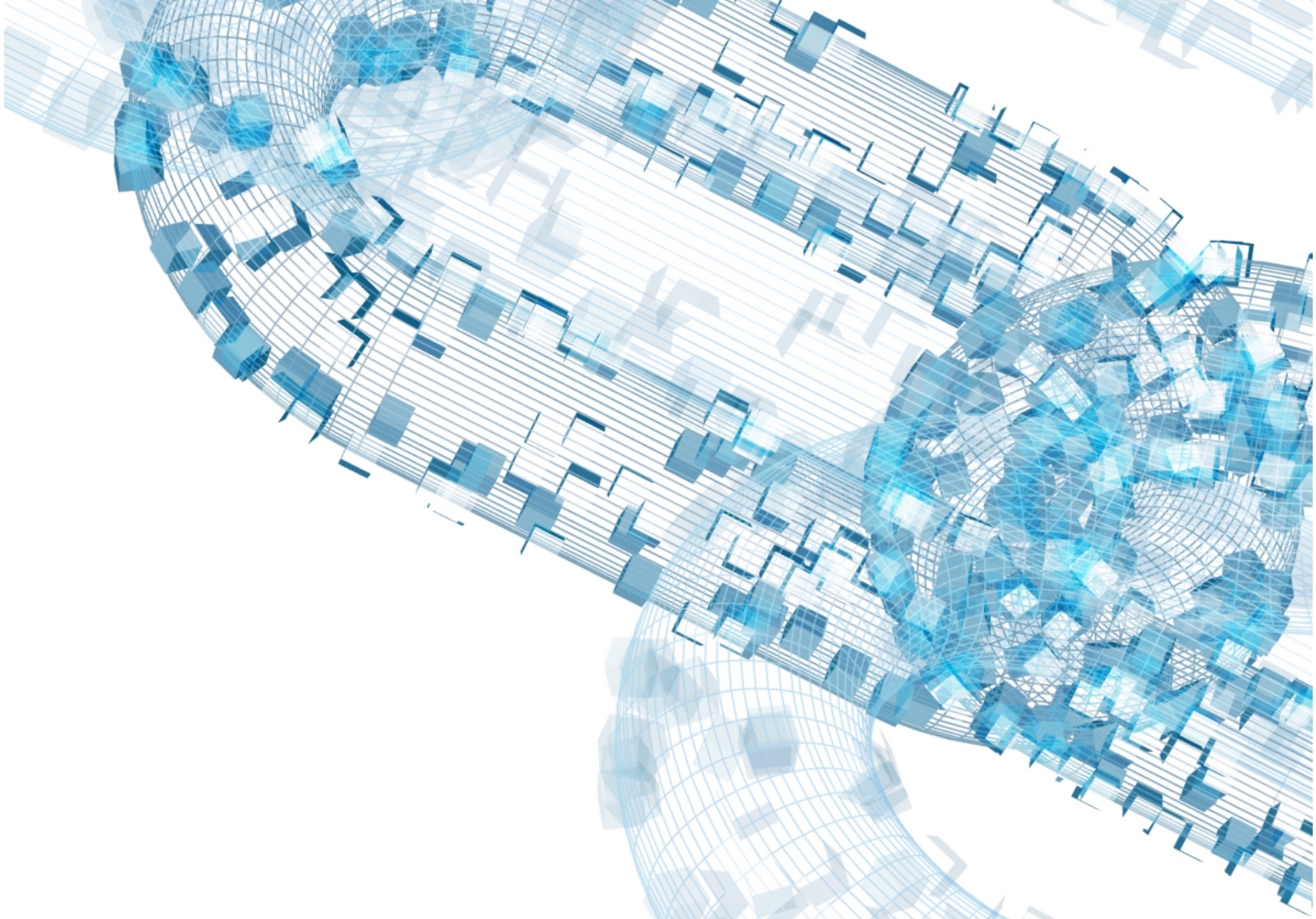

HYRA NETWORK

White Paper

A Decentralized and Verifiable AI Compute Mesh Leveraging Zero-Knowledge Proofs

Hyra Network: A Decentralized and Verifiable AI Compute Mesh Leveraging Zero-Knowledge Proofs

Hyra Network Team

September 30, 2025

1 Executive Summary

"Hyra Network: Building the Public AI Infrastructure of the Future"

Hyra Network is a decentralized, verifiable, and sovereign AI infrastructure protocol built as a Layer-3 blockchain on top of Ethereum and modular Layer-2 networks. It is designed to democratize access to AI computation and ensure cryptographic transparency for model outputs in a world increasingly driven by machine intelligence.

1.1 The Problem We Solve

Today's AI infrastructure is monopolized by hyperscale cloud providers. It is:

- **Centralized** - opaque, rent-seeking, and vulnerable to censorship.
- **Unverifiable** - users must "trust" model outputs blindly.
- **Inaccessible** - emerging markets, governments, and independent developers lack affordable and sovereign compute.
- **Unaccountable** - no public oversight, no community participation.

1.2 Our Vision

Hyra aims to become the foundational public utility for AI - verifiable, transparent, permissionless, and community-owned. We envision a world where:

- Citizens own the infrastructure that powers the intelligence around them.
- Developers deploy verifiable AI models through a global compute mesh.
- Governments run public-sector AI agents without depending on corporate clouds.
- Communities govern and evolve the protocol through an on-chain DAO.

1.3 What We've Built

Hyra combines:

1. **ZKML Runtime Layer** - Compiles AI models into circuits and generates ZK-STARK proofs of correctness. Ensures every model inference can be independently verified.
2. **Proof-of-Useful-Work Consensus** - Replaces meaningless hashing with real AI tasks. Stakers and nodes earn HYRA tokens by completing and proving inference.

3. **Global Compute Mesh (DePIN)** - Includes edge devices, GPU nodes, and sovereign cloud partners. Anyone can contribute compute and earn rewards in a permissionless manner.
4. **DAO Governance and Treasury** - Hyra Foundation (Switzerland) facilitates decentralized protocol governance. Rewards, grants, emissions, upgrades - all governed by DAO proposals. DAO members can opt into tokenized equity (STO) in Hyra Tek UAE, the commercial operator.

1.4 Token and Economic Model

- **Fixed supply:** 50 billion HYRA.
- **Distributed via:**
  - Staking rewards (60%)
  - Treasury and liquidity (12%)
  - Ecosystem growth (10%)
  - Team, advisors, strategic (18% total)
- **Reward Floor:** Tasks are rewarded in HYRA tokens with a floor pegged in USD ($0.01-$0.20/task).
- **Emission control:** GDP-based minting and burn unused supply.

1.5 Legal and Execution Structure

- **DAO Layer**: Hyra Foundation (Switzerland).
- **STO Layer**: Hyra Tek UAE issues tokenized equity.
- **IPO Layer**: Hyra Tek targets listing on Nasdaq / ADX by 2030.
- **Community contributors** can swap into equity via a DAO-to-STO bridge (KYC-light).

1.6 Roadmap Highlights (2025-2030)

Year	Milestones
2025	DAO launch, Reward Floor, staking live, SDK v1
2026	SubDAO framework, regional node mesh, grants DAO
2027	ZKML optimization, STO round 1
2028	Public SubDAO onboarding, GovMesh city partners
2029	IPO preparation and equity tokenization
2030	Public listing - DAO remains protocol steward

1.7 Open to All

Hyra is designed to be:

- **Permissionless** - anyone can run a node or build an AI agent.
- **Verifiable** - all compute outputs come with ZK-proof.
- **Public** - infrastructure built by and for the people.
- **DAO-governed** - fully transparent and upgradeable on-chain.

1.8 Why Hyra Matters

As AI becomes critical infrastructure, society must demand transparency, ownership, and sovereignty. Hyra is not just another blockchain - it is the AI layer for digital civilization. It is the infrastructure that ensures no AI acts unchecked - and no intelligence belongs to a few.

2 Problem and Vision

2.1 Abstract

This paper presents Hyra Network, a Layer-3 blockchain protocol leveraging zero-knowledge proofs to create a decentralized, scalable, and verifiable AI compute infrastructure. We develop a rigorous mathematical framework to build a global compute network encompassing 10^{10} edge and IoT devices across 200 countries. Our architecture combines ZK-STARKs, a Proof of Useful Work (PoUW) consensus mechanism, low-latency task orchestration (¡50ms), and a sophisticated balanced incentive system to create a decentralized network capable of competing with centralized cloud providers while providing transparency and cryptographically verifiable correctness. Our detailed mathematical analysis demonstrates the efficiency, scalability, and security of the proposed architecture, supporting Hyra Network's potential to revolutionize global AI infrastructure.

2.2 Introduction

The rapid advancement of artificial intelligence (AI) applications has led to an increasing demand for high-performance computing resources. Current systems, primarily relying on centralized server clusters, face significant challenges in scalability, cost, and transparency [1]. In this paper, we introduce Hyra Network, a breakthrough Layer-3 blockchain architecture designed to build a decentralized, scalable, and verifiable AI compute infrastructure.

2.3 Motivation and Challenges

The current centralized model for AI infrastructure faces several limitations:

- **Resource bottlenecks:** The growing demand for high-performance computing hardware, particularly GPUs, has led to severe shortages, limiting access to AI resources [2].
- **Economic barriers:** The costs of training and deploying advanced AI models have become prohibitively expensive for many organizations and individuals, leading to a concentration of power among a few large technology companies [3].
- **Lack of transparency and verifiability:** In centralized systems, users have no way to independently verify that AI computations are performed correctly without errors or fraud [4].
- **Sustainability:** Centralized AI data centers consume enormous amounts of electricity, raising serious concerns about environmental impact [5].

2.4 Our Contributions

This paper presents the following key contributions:

1. **DePAI (Decentralized Physical AI Infrastructure) Architecture:** We introduce a novel model combining principles of Decentralized Physical Infrastructure Networks (DePIN) with the specific requirements of AI computation.
2. **Zero-Knowledge Machine Learning (ZKML) Framework:** We develop a mathematical framework to represent AI algorithms as arithmetic circuits and generate ZK-STARK proofs allowing verification of both training and inference correctness.

3. **Low-Latency Task Orchestration:** We design a multi-criteria optimization algorithm to distribute AI tasks to the most suitable compute nodes with sub-50ms latency.

4. **Proof of Useful Work (PoUW) Consensus Mechanism:** We define a novel consensus mechanism based on useful AI computation, integrating blockchain security features with decentralized compute capability.

5. **Rigorous Mathematical Analysis:** We provide mathematical models to analyze the performance, security, and economics of Hyra Network, with detailed equations and proofs.

3 Background and Foundational Technologies

3.1 Decentralized Physical Infrastructure Networks (DePIN)

Decentralized Physical Infrastructure Networks (DePIN) represent a model where physical resources are contributed by distributed participants and coordinated through blockchain-based incentive mechanisms [6]. Typical DePIN systems include Filecoin for decentralized storage and Helium for wireless connectivity [7].

DePIN systems operate on the principle:

$$U_i = \mathbb{E}[\mathcal{R}_i] - C_i$$

Where:

- $\mathcal{U}_i$ is the net utility of participant i
- $\mathbb{E}[\mathcal{R}_i]$ is the expected reward
- $\mathcal{C}_i$ is the cost of participation

For the system to function effectively, incentive mechanisms must be designed such that $\mathcal{U}_i > 0$ for all honest participants.

3.2 Zero-Knowledge Proofs and ZK-STARKs

Zero-Knowledge Proofs (ZKPs) are cryptographic protocols allowing one party (the prover) to convince another party (the verifier) that a statement is true without revealing any additional information [8]. ZK-STARKs (Zero-Knowledge Scalable Transparent Arguments of Knowledge) are a type of ZKP offering the following properties [9]:

- **Zero-Knowledge:** Revealing no information beyond the correctness of the statement.
- **Computational Soundness:** A dishonest prover cannot create false proof.
- **Transparency:** No trusted setup required.
- **Scalability:** Verification cost increases logarithmically with computation size.
- **Post-Quantum Security:** Resistance against attacks from quantum computers.

A ZK-STARK for a program P with input x and output y is formally defined by the tuple of algorithms $(Gen, Prove, Verify)$:

- $Gen(1^\lambda, P) \to pp$: Generates public parameters pp with security parameter λ.
- $Prove(pp, x, w) \to \pi$: Generates a proof π for the statement "there exists a witness w such that $P(x, w) = y$".
- $Verify(pp, x, y, \pi) \to \{0, 1\}$: Verifies the proof π.

The security properties are formally defined as:

1. **Completeness:** $\Pr[\text{Verify}(pp, x, P(x, w), \text{Prove}(pp, x, w)) = 1] = 1$
2. **Soundness:** For any dishonest prover A, the following probability is negligible:

$$\Pr[\text{Verify}(pp, x, y, \pi) = 1 \wedge P(x, w) \neq y : \pi \leftarrow A(pp, x)]$$

3. **Zero-Knowledge:** There exists an efficient simulator S such that for any distinguishing verifier V:

$$|\Pr[V(pp, x, y, \pi) = 1 : \pi \leftarrow \text{Prove}(pp, x, w)] - \Pr[V(pp, x, y, \pi) = 1 : \pi \leftarrow S(pp, x, y)]| \leq \text{negl}(\lambda)$$

3.3 Layer-3 Blockchains and Multi-Layered Architecture

Layer-3 blockchains are protocols built on top of Layer-2 solutions, extending the capabilities of the underlying system [10]. The multi-layered blockchain architecture can be described as follows:

1. **Layer-1 (L1):** The base blockchain provides security and censorship resistance, but often limited in scalability and high in cost.
2. **Layer-2 (L2):** Scaling solutions like rollups that operate in parallel with L1, improving throughput and reducing costs, while inheriting security from L1.
3. **Layer-3 (L3):** Application-specific protocols built on L2, providing specialized functionality like AI computation or asset trading.

In this model, L1 serves as a settlement layer, L2 as a scaling layer, and L3 as an application layer. The mathematical model for transaction throughput of each layer is:

$$T_i = \frac{B_i}{t_i \cdot S_{tx}}$$

Where:

- T_i is the transaction throughput of layer i (TPS)
- B_i is the block size of layer i
- t_i is the block time of layer i
- S_{tx} is the average transaction size

With multi-layered architecture, we can achieve:

$$T_{L3} \gg T_{L2} \gg T_{L1}$$

4 Hyra Network Architecture

4.1 System Architecture Overview

Hyra Network is designed as a Layer-3 blockchain protocol specialized for decentralized AI computation. The overall architecture is illustrated in Figure 1 and includes the following components:

4.2 DePAI Architecture Components

The DePAI (Decentralized Physical AI Infrastructure) architecture of Hyra Network includes the following key components:

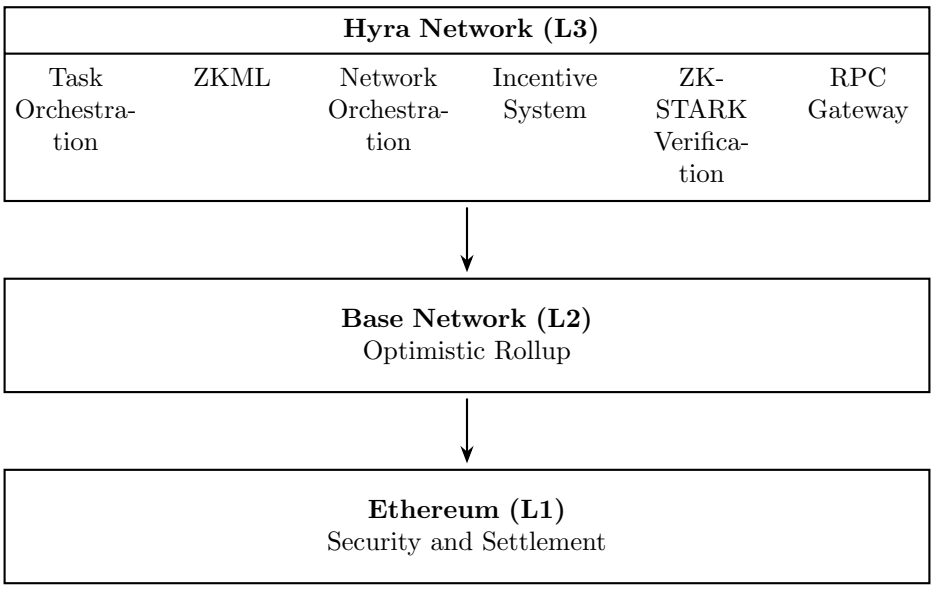

Figure 1: Overall Architecture of Hyra Network

4.2.1 Task Orchestrator

The task orchestrator is the central component responsible for distributing AI tasks to the most suitable compute nodes. The task allocation process is modeled as a multi-criteria optimization problem:

$$A^*(t) = \underset{n \in \mathcal{N}}{\operatorname{argmin}} \left[\alpha \cdot d(t,n) + \beta \cdot \frac{C_{\text{req}}(t)}{C_{\text{avail}}(n)} + \gamma \cdot \frac{1}{R(n)} + \delta \cdot L(n,t) \right]$$

Where:

- $A^*(t)$ is the optimal assignment for task t
- $\mathcal{N}$ is the set of all available nodes
- $d(t,n)$ is the geographical distance between the task source and node n
- $C_{\text{req}}(t)$ is the computational requirement of task t
- $C_{\text{avail}}(n)$ is the available computational capacity of node n
- $R(n)$ is the reputation rating of node n
- $L(n,t)$ is the estimated latency based on historical interaction between node n and task type t
- $\alpha, \beta, \gamma, \delta$ are dynamically adjusted weight coefficients

The geographical distance $d(t,n)$ is calculated using the Haversine formula:

$$d(t,n) = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_n - \phi_t}{2}\right) + \cos(\phi_t)\cos(\phi_n)\sin^2\left(\frac{\lambda_n - \lambda_t}{2}\right)}\right)$$

Where:

- r is the Earth's radius (≈ 6371 km)
- ϕ_t, λ_t are the latitude and longitude of the task source
- ϕ_n, λ_n are the latitude and longitude of the node

4.2.2 Node Network Management

Hyra Network manages a global network of up to 10^{10} edge and IoT devices. This network is modeled as a dynamic graph:

$$G(t) = (V(t), E(t))$$

Where:

- $V(t)$ is the set of active nodes at time t
- $E(t)$ is the set of connections between nodes at time t

Each node $v \in V(t)$ is characterized by an attribute vector:

$$a_v = (C_v, M_v, S_v, B_v, L_v, R_v, \phi_v, \lambda_v)$$

Where:

- C_v is the computational capacity (FLOPS for CPU, TFLOPS for GPU)
- M_v is the available memory (GB)
- S_v is the storage capacity (GB)
- B_v is the network bandwidth (Mbps)
- L_v is the average latency (ms)
- R_v is the reputation rating
- ϕ_v, λ_v are the latitude and longitude

The network topology is continuously optimized to minimize overall communication latency:

$$E^*(t) = \underset{E \subset V(t) \times V(t)}{\operatorname{argmin}} \sum_{(u,v) \in E} L(u, v)$$

with the constraint that the graph $G(t) = (V(t), E^*(t))$ remains connected.

4.2.3 Reputation System

Each node in Hyra Network is assigned a reputation rating, which is continuously updated based on its performance. We use a modified version of the Elo rating algorithm:

$$R'_i = R_i + K \cdot (S_i - E_i)$$

Where:

- R'_i is the new rating of node i
- R_i is the current rating
- K is the adjustment factor
- S_i is the actual score (1 for success, 0 for failure)
- E_i is the expected score, calculated as:

$$E_i = \frac{1}{1 + 10^{(R_b - R_i)/400}}$$

Where R_b is the benchmark rating for the task type.

The K-factor is dynamically adjusted based on task complexity and node history:

$$K = K_{\text{base}} \cdot \sqrt{\frac{C_t}{C_{\text{avg}}}} \cdot \left(1 + \frac{\ln(N_t + 1)}{\ln(100)}\right)$$

Where:

- K_{base} is the base K-factor (default = 32)
- C_t is the computational complexity of the task
- C_{avg} is the average computational complexity of tasks in the network
- N_t is the number of tasks completed by the node

5 Zero-Knowledge Machine Learning (ZKML) Framework

5.1 Theoretical Foundation

Zero-Knowledge Machine Learning (ZKML) is the fusion of machine learning with zero-knowledge proofs to create verifiable AI computations [11]. In Hyra Network, we develop a comprehensive ZKML framework using ZK-STARKs.

The ZKML process can be divided into the following stages:

1. **Computation Representation:** Converting an AI algorithm into an arithmetic circuit.
2. **Proof Generation:** Creating a ZK-STARK proof for the correct execution of the circuit.
3. **Proof Verification:** Verifying the correctness of the proof without re-executing the full computation.

5.2 Arithmetic Representation of AI Algorithms

To apply ZK-STARKs to AI computation, we need to represent AI algorithms as arithmetic circuits over a finite field $\mathbb{F}_p$ where p is a large prime.

An arithmetic circuit C over $\mathbb{F}_p$ is a directed acyclic graph (DAG) where:

- Each node represents an arithmetic operation (+, -, $\times$, $\div$)
- Each edge represents data flow between operations
- Inputs and outputs of each node are elements of $\mathbb{F}_p$

For a neural network model f_θ with parameters θ, we represent the inference process as an arithmetic circuit C_{f_θ}:

$$C_{f_\theta} : \mathbb{F}_p^n \to \mathbb{F}_p^m$$

Where n is the input dimension and m is the output dimension.

Common AI operations are represented as follows:

1. **Matrix Multiplication:** For a weight matrix $W \in \mathbb{F}_p^{m \times n}$ and input vector $x \in \mathbb{F}_p^n$, the matrix multiplication $y = Wx$ is represented using $m \times n$ multiplications and $m \times (n-1)$ additions in $\mathbb{F}_p$.
2. **Activation Functions:** Non-linear functions like ReLU, sigmoid, and tanh are approximated by polynomials over $\mathbb{F}_p$. For example, sigmoids can be approximated by:

$$\sigma(x) \approx \sum_{i=0}^{d} a_i x^i$$

where coefficients a_i are optimized to minimize approximation error over a finite range.

3. **Pooling and Normalization:** These operations are represented using basic arithmetic operations (addition, multiplication, division) and comparisons.

Table 1 presents the computational cost (number of arithmetic operations) of common neural network layers.

Table 1: Computational Cost of Neural Network Layers

Layer Type	**Core Operation**	**Cost (Number of Operations)**
Fully Connected ($n \times m$)	Matrix multiplication	$O(n \cdot m)$
Convolutional ($k \times k, c_{in}, c_{out}$)	Convolution	$O(k^2 \cdot c_{in} \cdot c_{out} \cdot h \cdot w)$
ReLU	$\max(0, x)$	$O(n)$
Sigmoid (degree-d approx.)	Polynomial evaluation	$O(d \cdot n)$
BatchNorm	Normalization	$O(n)$
MaxPooling ($k \times k$)	Max selection	$O(k^2 \cdot c \cdot h \cdot w)$

5.3 ZK-STARK Proof Generation and Verification

To generate a ZK-STARK proof for an AI computation, we use the following approach:

1. **Polynomial Representation:** Convert the arithmetic circuit C into a set of polynomial constraints $\{p_i(x) = 0\}_{i=1}^{m}$.
2. **Reed-Solomon Encoding:** Encode the polynomials as Reed-Solomon codewords to allow efficient checking.
3. **Polynomial Commitments:** Create cryptographic commitments to the polynomials.
4. **FRI Protocol:** Use the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI) protocol to verify proximity of polynomials to Reed-Solomon codewords.

The computational costs of proof generation and verification are summarized in Table 2.

Table 2: Computational Costs of the ZK-STARK Proof System

Phase	**Time Complexity**	**Space Complexity**
Proof Generation	$O(N \log N)$	$O(N)$
Proof Verification	$O(\log^2 N)$	$O(\log^2 N)$

Where N is the size of the arithmetic circuit (number of operations).

5.4 zkLLVM for AI Compilation Optimization

To simplify the process of creating arithmetic circuits from AI source code, we develop zkLLVM, a specialized compiler that transforms AI source code (C++, Python) into optimized arithmetic circuit representations [12].

The zkLLVM compilation pipeline includes the following steps:

1. **Front-end:** Parse source code and generate intermediate representation (IR).
2. **Middle-end:** Optimize IR by eliminating dead code, simplifying expressions, and applying AI-specific transformations.
3. **Back-end:** Generate optimized arithmetic circuits from IR.

AI-specific optimizations in zkLLVM include:

- **Vectorization:** Efficient representation of vector and matrix operations.

- **Constant folding:** Pre-computing expressions with constants.
- **Common subexpression elimination:** Reusing results of repeated computations.
- **Loop unrolling:** Expanding loops to allow further optimization.

6 Decentralized AI Compute Mesh

6.1 Network Capacity Model

Hyra Network aggregates the computational power of 10^{10} edge and IoT devices globally. The total computational capacity is modeled as:

$$C_{\text{total}} = \sum_{i=1}^{n} w_i \cdot c_i \cdot a_i \cdot e_i$$

Where:

- n is the number of participating devices
- c_i is the computational capacity of device i (FLOPS for CPU, TFLOPS for GPU)
- a_i is the availability ratio of device i (proportion of time the device is online)
- e_i is the computational efficiency of device i (ratio of theoretical capacity achieved in practice)
- w_i is an efficiency factor based on hardware type and compatibility with AI tasks

Based on global device distribution and specifications, we estimate the overall network capacity (Table 3).

Table 3: Estimated Computational Capacity of Hyra Network

Device Type	Count	Avg. Capacity	Availability	Efficiency	Total Capacity
Smartphones	5×10^9	5 TFLOPS	0.3	0.6	4.5×10^{10} TFLOPS
Personal Computers	2×10^9	20 TFLOPS	0.5	0.7	1.4×10^{11} TFLOPS
IoT Devices	3×10^9	0.5 TFLOPS	0.8	0.5	6.0×10^9 TFLOPS
Edge Servers	5×10^6	100 TFLOPS	0.9	0.8	3.6×10^9 TFLOPS
Total	$\mathbf{1 \times 10^{10}}$	-	-	-	$\mathbf{\approx 1.9 \times 10^{11}}$ **TFLOPS**

With an estimated total capacity of approximately 1.9×10^{11} TFLOPS, Hyra Network has the potential to provide computational power equivalent to about 1.9 million NVIDIA A100 GPUs (100 TFLOPS per GPU).

6.2 Low-Latency Task Management

Hyra Network employs a hierarchical latency model to estimate and optimize the overall system latency:

$$L_{\text{total}} = L_{\text{alloc}} + L_{\text{data}} + L_{\text{exec}} + L_{\text{proof}} + L_{\text{verify}}$$

Where:

- L_{alloc} is the task allocation latency
- L_{data} is the data transfer latency
- L_{exec} is the task execution latency
- L_{proof} is the proof generation latency

- L_{verify} is the proof verification latency

To achieve sub-50ms task allocation latency, we use a hierarchical geospatial indexing technique:

1. **Global Partitioning:** Divide the Earth into an H3 grid of resolution 8, creating approximately 6.4 million grid cells.
2. **Spatial Indexing:** Each node is assigned to a grid cell based on its geographical location.
3. **Approximate Nearest Neighbor Search:** When a new task is submitted, we search for the most suitable nodes in the grid cell containing the task source and neighboring cells.

The time complexity of the task allocation algorithm is:

$$T_{\text{alloc}} = O(1) + O(\log k)$$

Where k is the number of nodes in the considered grid cells. The $O(1)$ component corresponds to grid cell lookup, and $O(\log k)$ corresponds to finding the optimal node within those cells.

6.3 Fault Tolerance and Resilience

Hyra Network designs a multi-layered fault tolerance model to ensure high reliability even when a significant proportion of nodes are unavailable or unreliable:

1. **N+k Redundancy:** Each task is assigned to N+k nodes, with k backup nodes. The probability of successful task completion is:

   $$P_{\text{success}} = 1 - \binom{N+k}{N} \cdot (1-p)^{N+k}$$

   Where p is the probability of an individual node successfully completing the task.

2. **Time Distribution:** The distribution of task completion time can be modeled using a Weibull distribution:

   $$f(t;\lambda,k) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-(t/\lambda)^k}$$

   Where λ is the scale parameter and k is the shape parameter.

3. **Time-Based Reliability Model:** The reliability of the network over time is modeled using an exponential function:

   $$R(t) = e^{-\lambda t}$$

   Where λ is the failure rate.

Table 4: System Reliability with Different Redundancy Configurations

Redundancy Config.	Node Success Probability	System Reliability
N=1, k=0	0.9	0.9
N=1, k=1	0.9	0.99
N=1, k=2	0.9	0.999
N=2, k=1	0.9	0.999
N=3, k=2	0.9	0.99999

7 Incentive Mechanism and Reputation System

7.1 AI-to-Earn Model and Reward Calculation

Hyra Network employs an AI-to-Earn model to incentivize users to contribute computational resources. The reward for completing an AI task is calculated as:

$$R_t = B_t \cdot \frac{C_{\text{node}}}{C_{\text{required}}} \cdot \Delta T_t \cdot S_{\text{node}} \cdot P_{\text{node}} \cdot Q_{\text{result}}$$

Where:

- B_t is the base reward for the AI task type
- C_{node} is the computational resource contributed by the node
- C_{required} is the minimum computational resources required for the task
- ΔT_t is the actual execution time
- S_{node} is the stake factor, proportional to the amount of HYRA tokens staked
- P_{node} is the performance factor based on the node's Elo rating
- Q_{result} is the quality factor of the result (1 for correct, 0 for incorrect)

The stake factor is defined as:

$$S_{\text{node}} = \min\left(1, \sqrt{\frac{\text{HYRA}_{\text{staked}}}{\text{HYRA}_{\text{baseline}}}}\right)$$

Where $\text{HYRA}_{\text{baseline}}$ is the baseline token amount to achieve maximum stake factor. The performance factor P_{node} is a function of the node's Elo rating:

$$P_{\text{node}} = \frac{1}{1 + e^{-\alpha(R_{\text{node}} - R_{\min})}}$$

Where α is a scaling factor and $R_{\min}$ is the minimum rating.

7.2 Game-Theoretic Analysis and Nash Equilibrium

We analyze Hyra Network's incentive mechanism from a game-theoretic perspective to ensure that nodes are incentivized to act honestly. Consider an infinitely repeated game $\Gamma = (N, A, u)$ where:

- N is the set of players (nodes)
- A is the set of actions (honest or cheat)
- u is the utility function

The expected utility for a node following strategy s is:

$$u(s) = \sum_{t=0}^{\infty} \delta^t \cdot r_t(s)$$

Where:

- δ is the discount factor
- $r_t(s)$ is the reward at time t when following strategy s

We prove that under reasonable conditions, the honest strategy always forms a subgame-perfect Nash equilibrium for the repeated game, meaning no node has an incentive to cheat.

Theorem 7.1. *Assuming $\delta > 1 - \frac{r_{honest}}{r_{cheat}}$, where r_{honest} is the reward for acting honestly and r_{cheat} is the reward for cheating in a single round. Then, the honest strategy is a subgame-perfect Nash equilibrium of the infinitely repeated game.*

Proof. Consider a node contemplating cheating. If this node cheats, it may receive a reward r_{cheat} in the current round but will be penalized in future rounds (e.g., reputation rating decrease or exclusion from the network). The utility of cheating is:

$$u_{\text{cheat}} = r_{\text{cheat}} + \sum_{t=1}^{\infty} \delta^t \cdot r_{\text{punish}} = r_{\text{cheat}} + \frac{\delta}{1-\delta} \cdot r_{\text{punish}}$$

Where r_{punish} is the reward after being penalized. If the node acts honestly, its utility is:

$$u_{\text{honest}} = r_{\text{honest}} + \sum_{t=1}^{\infty} \delta^t \cdot r_{\text{honest}} = r_{\text{honest}} + \frac{\delta}{1-\delta} \cdot r_{\text{honest}} = \frac{r_{\text{honest}}}{1-\delta}$$

For $u_{\text{honest}} > u_{\text{cheat}}$, we need:

$$\frac{r_{\text{honest}}}{1-\delta} > r_{\text{cheat}} + \frac{\delta}{1-\delta} \cdot r_{\text{punish}}$$

If $r_{\text{punish}} = 0$ (node is excluded from the network), the condition becomes:

$$\frac{r_{\text{honest}}}{1-\delta} > r_{\text{cheat}}$$

From which:

$$\delta > 1 - \frac{r_{\text{honest}}}{r_{\text{cheat}}}$$

□

7.3 Dynamic Elo Rating System

Hyra Network's Elo rating system is designed to reliably assess each node's capability and adjust quickly when this capability changes. The rating update equation is:

$$R_i' = R_i + K_i \cdot (S_i - E_i)$$

Where K_i is a dynamic K-factor calculated as:

$$K_i = K_{\text{base}} \cdot \sqrt{\frac{C_t}{C_{\text{avg}}}} \cdot \left(1 + \frac{\ln(N_i + 1)}{\ln(100)}\right) \cdot \sqrt{\frac{1500}{R_i}}$$

This formula includes the following components:

- $\sqrt{\frac{C_t}{C_{\text{avg}}}}$: Adjustment based on task complexity
- $\left(1 + \frac{\ln(N_i+1)}{\ln(100)}\right)$: Adjustment based on experience (number of tasks completed)
- $\sqrt{\frac{1500}{R_i}}$: Adjustment based on current rating (lower-rated nodes change faster)

The expected score E_i is calculated as:

$$E_i = \frac{1}{1 + 10^{(R_b - R_i)/400}}$$

Where R_b is the benchmark rating for the task type, calculated as:

$$R_b = 1500 + 100 \cdot \log_{10}\left(\frac{C_t}{C_{\text{min}}}\right)$$

Where C_t is the computational complexity of the task and C_{min} is the minimum computational complexity.

8 Blockchain Performance Analysis

8.1 Throughput and Scalability Model

The throughput of Hyra Network's blockchain (Layer-3) is modeled by the formula:

$$\text{TPS} = \min\left(\frac{B_{\text{size}}}{T_{\text{size}} \cdot T_{\text{block}}}, \frac{C_{\text{total}}}{C_{\text{tx}}}, \frac{1}{L_{\text{consensus}}}\right)$$

Where:

- B_{size} is the block size (bytes)
- T_{size} is the average transaction size (bytes)
- T_{block} is the average block time (seconds)
- C_{total} is the total processing capacity (computations/second)
- C_{tx} is the processing cost per transaction (computations)
- $L_{\text{consensus}}$ is the consensus latency (seconds)

With Hyra Network's design parameters:

- $B_{\text{size}} = 10$ MB
- $T_{\text{size}} = 500$ bytes
- $T_{\text{block}} = 2$ seconds
- $C_{\text{total}} \approx 10^{12}$ computations/second
- $C_{\text{tx}} \approx 10^{6}$ computations
- $L_{\text{consensus}} \approx 0.5$ seconds

This yields a theoretical TPS of approximately 10,000 transactions per second.

8.2 Gas Cost Model and Economic Efficiency

The gas cost for a transaction in Hyra Network is calculated as:

$$\text{Gas} = G_{\text{base}} + G_{\text{data}} \cdot S_{\text{data}} + G_{\text{compute}} \cdot C_{\text{tx}} + G_{\text{proof}} \cdot S_{\text{proof}}$$

Where:

- G_{base} is the base gas cost per transaction
- G_{data} is the gas cost per byte of data
- S_{data} is the data size (bytes)
- G_{compute} is the gas cost per computation unit
- C_{tx} is the number of computation units
- G_{proof} is the gas cost per byte of proof
- S_{proof} is the proof size (bytes)

Table 5 presents the gas costs for common transaction types in Hyra Network.

Table 5: Gas Costs for Transaction Types

Transaction Type	Base Gas	Data Size	Proof Size	Comp. Cost	Total Gas
Submit Task	21,000	1–10 KB	0	Low	∼30,000
Submit Proof	21,000	1–5 KB	20–100 KB	Medium	∼150,000
Verify Task	21,000	<1 KB	0	High	∼50,000
Claim Reward	21,000	<1 KB	0	Low	∼25,000

8.3 Confirmation Latency Model

The confirmation latency in Hyra Network is modeled as:

$$L_{\text{confirmation}} = T_{\text{block}} \cdot N_{\text{conf}} + L_{\text{prop}}$$

Where:

- T_{block} is the average block time
- N_{conf} is the number of confirmations required
- L_{prop} is the propagation latency

With $T_{\text{block}} = 2$ seconds, $N_{\text{conf}} = 1$ for regular transactions, and $L_{\text{prop}} \approx 200$ ms, the average confirmation latency is approximately 2.2 seconds.

9 Technical Implementation Details

9.1 Go Ethereum Layer-3 Architecture

Hyra Network is built on a customized version of Go Ethereum (geth), with significant modifications to support decentralized AI computation:

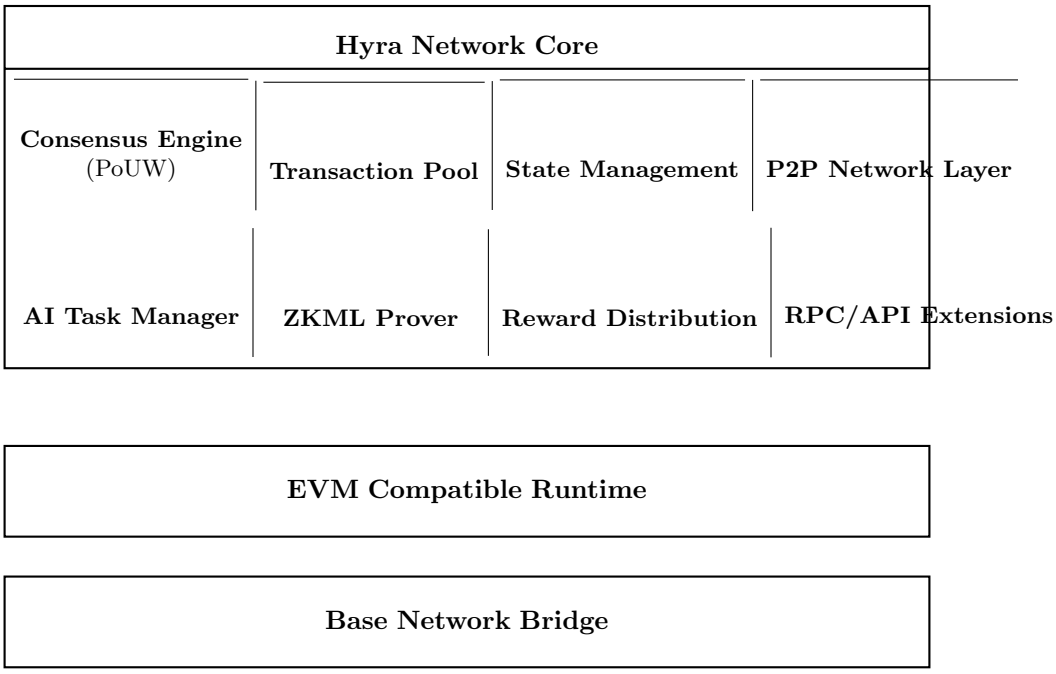

Figure 2: Detailed Architecture of Hyra Network Based on Go Ethereum

Key components include:

1. **Consensus Engine (PoUW):** Replaces the standard PoW/PoS consensus mechanism with Proof of Useful Work.
2. **AI Task Manager:** Manages the lifecycle of AI tasks, from submission to completion.
3. **ZKML Prover:** Integrates with zkLLVM to generate ZK-STARK proofs for AI computations.
4. **Reward Distribution:** Manages the distribution of HYRA tokens to nodes based on their contributions.
5. **RPC/API Extensions:** Extends the standard API with AI-specific methods.

9.2 RPC Extensions for Task Management

Hyra Network extends Ethereum's standard JSON-RPC API with the following methods:

```
// Task Management
hyra_submitTask(taskType, input, config) -> taskID
hyra_getTask(taskID) -> {status, assignment, deadline}
hyra_cancelTask(taskID) -> boolean
// Proof Handling
hyra_submitProof(taskID, proof, result) -> {status, txHash}
hyra_verifyProof(taskID, proof, result) -> boolean
// Node Management
hyra_registerNode(nodeID, specs, location) -> {status, txHash}
hyra_getNodeStats(nodeID) -> {rating, rewards, tasks, availability}
hyra_updateNodeSpecs(nodeID, newSpecs) -> boolean
// Network Statistics
hyra_getNetworkStats() -> {activeNodes, pendingTasks, completedTasks,
    totalCompute}
hyra_getTaskMetrics(taskType) -> {avgCompletionTime, avgReward, successRate}
```

Listing 1: Hyra Network RPC Extensions

9.3 Cross-Platform ZK-STARK Library

Hyra Network's ZK-STARK library is implemented with AI-specific optimizations:

Hyra ZK-STARK Library			
Field Arithmetic	Polynomial Operations	Merkle Tree	FRI Protocol Implementation
Circuit Compiler	Constraint Generator	STARK Prover	STARK Verifier
Platform-Specific Optimizations			
x86-64 (AVX, SSE)	ARM (NEON)	WebAssembly (SIMD)	CUDA/OpenCL GPU Acceleration

Figure 3: ZK-STARK Library Architecture

The library is implemented in multiple programming languages and optimized for multiple platforms:

1. **Rust:** Core implementation with high performance and strong security.
2. **C/C++:** High-performance bindings for applications and embedded systems.
3. **WebAssembly:** Enables running in browsers and Node.js environments.

4. **Go:** Integration with Hyra Network nodes.

5. **Python:** Bindings for AI developers to easily integrate with frameworks like TensorFlow and PyTorch.

9.4 Smart Contracts for Verification and Rewards

Hyra Network implements the following smart contracts to manage proof verification and reward distribution:

9.4.1 TaskRegistry.sol

Manages the lifecycle of AI tasks.

```
contract TaskRegistry {
    struct Task {
        address submitter;
        uint256 taskType;
        bytes32 inputHash;
        uint256 reward;
        uint256 deadline;
        TaskStatus status;
        address[] assignedNodes;
        mapping(address => bool) proofSubmitted;
        mapping(address => bytes32) resultHash;
    }

    enum TaskStatus { Pending, Assigned, Completed, Cancelled }

    mapping(bytes32 => Task) public tasks;

    event TaskSubmitted(bytes32 indexed taskId, address indexed submitter, uint256
    reward);
    event TaskAssigned(bytes32 indexed taskId, address[] nodes);
    event TaskCompleted(bytes32 indexed taskId, address indexed node, bytes32
    resultHash);

    function submitTask(uint256 taskType, bytes calldata input, uint256 deadline)
    external payable returns (bytes32);
    function assignTask(bytes32 taskId, address[] calldata nodes) external;
    function submitProof(bytes32 taskId, bytes calldata proof, bytes32 resultHash)
    external;
    function cancelTask(bytes32 taskId) external;
}
```

Listing 2: TaskRegistry.sol Smart Contract

9.4.2 ProofVerifier.sol

Verifies ZK-STARK proofs.

```
contract ProofVerifier {
    struct VerificationKey {
        uint256 domainSize;
        bytes32 constraintHash;
        uint256 fieldModulus;
        // Additional parameters...
    }

    mapping(uint256 => VerificationKey) public verificationKeys;

    event ProofVerified(bytes32 indexed taskId, address indexed prover, bool success);

    function verifyProof(
        uint256 taskType,
```

```
        bytes32 inputHash,
        bytes32 outputHash,
        bytes calldata proof
    ) external returns (bool);

    function registerVerificationKey(uint256 taskType, VerificationKey calldata key)
    external;
}
```

Listing 3: ProofVerifier.sol Smart Contract

9.4.3 RewardDistributor.sol

Manages reward distribution to nodes.

```
contract RewardDistributor {
    struct NodeInfo {
        uint256 rating;
        uint256 computePower;
        uint256 stakedAmount;
        uint256 completedTasks;
        uint256 totalRewards;
    }

    mapping(address => NodeInfo) public nodes;

    event RewardDistributed(address indexed node, uint256 amount, bytes32 indexed
    taskId);
    event RatingUpdated(address indexed node, uint256 newRating);

    function distributeReward(bytes32 taskId, address node) external;
    function updateRating(address node, uint256 newRating) external;
    function stake(uint256 amount) external;
    function unstake(uint256 amount) external;
}
```

Listing 4: RewardDistributor.sol Smart Contract

10 Proof of Useful Work (PoUW) Consensus Mechanism

10.1 Consensus Mechanism Design

Hyra Network's Proof of Useful Work (PoUW) consensus mechanism redefines the concept of "work" in blockchain by focusing on useful AI computation. Instead of solving meaningless hash puzzles as in traditional Proof of Work, PoUW nodes compete to complete real AI tasks and generate ZK-STARK proofs.

The probability of a node generating a block is proportional to the amount of useful AI work it performs:

$$P(\text{block} \mid \text{node}_i) = \frac{W_i}{\sum_j W_j}$$

Where W_i is the amount of useful work performed by node i, defined as:

$$W_i = \sum_{t \in T_i} C_t \cdot V_t \cdot Q_t$$

Where T_i is the set of tasks completed by node i, C_t is the computational complexity of task t, V_t is the value of that task to the network, and Q_t is the quality of the result (verified by the ZK-STARK proof).

10.2 Difficulty Adjustment and Energy Consumption

The difficulty of PoUW is dynamically adjusted to maintain stable block times:

$$D_{\text{new}} = D_{\text{current}} \cdot \frac{T_{\text{target}}}{T_{\text{actual}}}$$

Where D is the difficulty, T_{target} is the target block time, and T_{actual} is the recent actual block time.

Compared to traditional Proof of Work, PoUW saves significant energy because computational work is used for useful purposes. Energy efficiency can be expressed as:

$$E_{\text{PoUW}} = \frac{W_{\text{useful}}}{W_{\text{total}}} \cdot E_{\text{PoW}}$$

Where E_{PoUW} and E_{PoW} are the energy consumption of PoUW and PoW respectively, W_{useful} is the useful work, and W_{total} is the total work.

10.3 Attack Resistance and Security

The PoUW mechanism faces unique attack vectors not encountered in traditional consensus mechanisms. We analyze attack vectors and mitigations:

1. **Fake Result Attack:** A node might try to claim rewards without actually performing computation.
   - *Mitigation:* ZK-STARK proofs ensure that computation is performed correctly.
2. **Selfish Mining:** A node might only include transactions beneficial to it.
   - *Mitigation:* Incentive mechanisms ensure that including all valid transactions is optimal.
3. **Work Reuse Attack:** A node might try to claim multiple rewards for the same computation.
   - *Mitigation:* Each task has a unique ID and can only be completed once.
4. **Utility Splitting Attack:** A node might try to split a large task into smaller ones to maximize rewards.
   - *Mitigation:* Reward structure is designed to balance against splitting costs.

Table 6 provides a summary of potential attacks on the PoUW mechanism and their respective mitigations.

Table 6: Potential Attacks on PoUW and Mitigations

Attack Type	Description	Mitigation	Effectiveness
Fake Result	Claiming rewards without performing computation	Mandatory ZK-STARKs	High
Selfish Mining	Only including beneficial transactions	Incentive mechanisms	Medium
Work Reuse	Claiming multiple rewards for same computation	Unique task IDs	High
Utility Splitting	Splitting tasks to maximize rewards	Reward structure	Medium
Denial of Service	Accepting tasks but not completing them	Reputation system	High

11 Security Considerations

11.1 Security Model

Hyra Network employs a multi-layered security model to protect the integrity and privacy of the system:

1. **Cryptographic Security:** ZK-STARKs ensure that computations are verified without revealing underlying data.
2. **Economic Security:** Staking and reputation models incentivize honest behavior. The probability of a successful attack is bounded by:

$$P_{\text{attack}} \leq \min\left(1, \frac{R_{\text{attack}}}{C_{\text{attack}} + S_{\text{lost}}}\right)$$

   Where R_{attack} is the reward from attacking, C_{attack} is the cost of attack, and S_{lost} is the stake value lost.
3. **Network Security:** P2P protocol is enhanced with end-to-end encryption and node authentication based on digital signatures.
4. **Data Security:** Sensitive data is encrypted during transmission and storage.

11.2 Vulnerability Analysis

We analyze potential vulnerabilities of Hyra Network and their mitigations:

1. **Sybil Attack:** An attacker might create multiple nodes to increase reward share.
   - *Analysis:* The net utility of a Sybil attack is:

$$U_{\text{sybil}} = n \cdot r - c \cdot n - S_{\text{initial}}$$

     Where n is the number of nodes, r is the average reward, c is the operating cost per node, and S_{initial} is the initialization cost.
   - *Mitigation:* Staking requirements and reputation system make Sybil attacks economically inefficient.
2. **Eclipse Attack:** Isolating a node from the network by controlling all of its connections.
   - *Analysis:* The probability of success for an Eclipse attack is:

$$P_{\text{eclipse}} = \left(\frac{n_{\text{adversary}}}{n_{\text{total}}}\right)^k$$

     Where $n_{\text{adversary}}$ is the number of malicious nodes, n_{total} is the total number of nodes, and k is the number of connections for the target node.
   - *Mitigation:* Maintaining a diverse and random set of connections.
3. **51% Attack:** Controlling the majority of computational power to control the network.
   - *Analysis:* The cost of a 51% attack is:

$$C_{51\%} > \frac{C_{\text{total}}}{2}$$

     Where C_{total} is the total computational capacity of the network.
   - *Mitigation:* Globally distributed networks with 10^{10} devices makes 51% attacks economically infeasible.

4. **DoS Attack:** Overwhelming the network with invalid requests.
    - *Analysis:* The effectiveness of a DoS attack depends on the attack bandwidth and network processing capacity:
    $$E_{\text{DoS}} = \min\left(1, \frac{B_{\text{attack}}}{B_{\text{network}}}\right)$$
    Where B_{attack} is the attack bandwidth and B_{network} is the network processing capacity.
    - *Mitigation:* DoS defenses including rate limiting, anomaly detection, and load distribution.

11.3 AI-Specific Attack Mitigations

Hyra Network also addresses AI-specific attacks:

1. **Data Poisoning:** Introducing malicious data to influence results.
    - *Mitigation:* Input data validation, consistency checks, and anomaly detection.
2. **Model Stealing:** Extracting model parameters through careful queries.
    - *Mitigation:* Query limiting, noise addition, and model obfuscation through ZK-STARKs.
3. **Adversarial Examples:** Specially crafted inputs designed to cause incorrect predictions.
    - *Mitigation:* Robust training, input validation, and adversarial input detection.

12 Cross-Chain Interoperability

12.1 Blockchain Bridge Architecture

Hyra Network implements a blockchain bridge architecture to enable seamless integration with multiple blockchains. The architecture is illustrated in Figure 4.

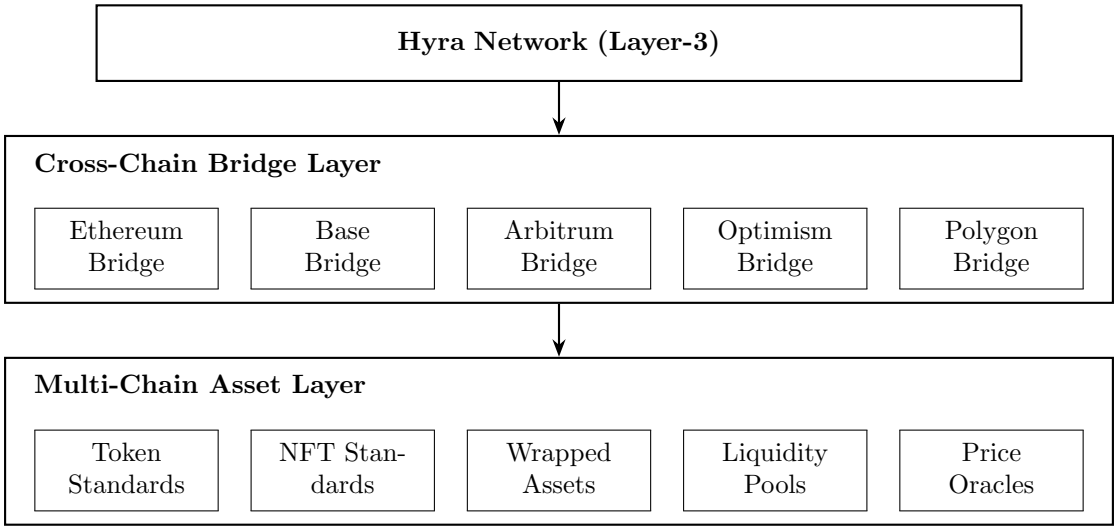

Figure 4: Hyra Network's Cross-Chain Bridge Architecture

The cross-chain bridge enables:

1. **Token Transfer:** Moving HYRA tokens and other assets between blockchains.
2. **Data Access:** Accessing data from other blockchains for use in AI computations.
3. **Cross-Chain Verification:** Verifying ZK-STARK proofs across multiple blockchains.

12.2 Cross-Chain Messaging Protocol

Hyra Network employs a cross-chain messaging protocol to facilitate communication between different blockchains:

$$M_{i \to j} = \text{Encrypt}(m, k_{i,j})$$

Where $M_{i \to j}$ is the encrypted message from chain i to chain j, m is the original message, and $k_{i,j}$ is the shared key between the two chains.

The message passing process includes:

1. **Packaging:** Message is packaged and encrypted on the source chain.
2. **Relaying:** Message is relayed by a trusted relay network.
3. **Verification:** Message is verified on the destination chain.
4. **Execution:** Action is taken based on the message on the destination chain.

12.3 Cross-Chain Proof Verification

To enable verification of ZK-STARK proofs across multiple blockchains, we use a distributed verification mechanism:

$$\text{Verify}(\pi, x, y) = \bigwedge_{i=1}^{n} \text{Verify}_i(\pi_i, x_i, y_i)$$

Where π_i, x_i, and y_i are portions of the proof, input, and output distributed to blockchain i.

This mechanism allows verification of large proofs that might exceed the gas limits of a single blockchain.

13 Future Research and Development

13.1 ZKML Improvements

Future research directions to improve ZKML include:

1. **Recursive STARKs:** Allowing multiple proofs to be aggregated into a single proof, significantly reducing on-chain verification costs.

$$\pi_{\text{combined}} = \text{ProveRecursive}(\pi_1, \pi_2, \ldots, \pi_n)$$

2. **Hardware-Accelerated Proving:** Developing dedicated hardware implementations to accelerate ZK-STARK proof generation.

$$T_{\text{prove}}^{\text{accelerated}} = T_{\text{prove}}^{\text{software}} / \alpha_{\text{acceleration}}$$

   Where $\alpha_{\text{acceleration}}$ is the acceleration factor, potentially reaching 100-1000x with dedicated hardware.

3. **Plonky3:** Combining advantages of different proof systems (STARKs, SNARKs, Bulletproofs) to create a hybrid proving system with lower costs and stronger security.

13.2 Compute Network Expansion

Plans to expand the computer network include:

1. **Trusted Execution Environments (TEEs):** Integration with trusted execution environments like Intel SGX, ARM TrustZone, and AMD SEV to provide additional security and privacy.

2. **Quantum-Resistant Cryptography:** Preparing for the quantum era by implementing post-quantum cryptographic algorithms.

3. **Federated Learning:** Implementing federated learning to train AI models on distributed data without sharing the raw data.

$$w_{t+1} = w_t - \eta \cdot \frac{1}{K} \sum_{k=1}^{K} \nabla F_k(w_t)$$

Where w_t is the model weights at time t, η is the learning rate, K is the number of clients, and $\nabla F_k(w_t)$ is the gradient of the loss function at client k.

13.3 Layer-3 Blockchain Advancements

Improvements in Layer-3 blockchain technology include:

1. **State Channels:** Implementing state channels to increase throughput and reduce latency for frequent interactions.

2. **Homomorphic Encryption:** Integrating homomorphic encryption to allow computation directly on encrypted data.

$$E(x) \cdot E(y) = E(x + y)$$

$$E(x)^y = E(x \cdot y)$$

3. **Adaptive Sharding:** Dynamically splitting the network into shards to improve scalability.

$$S = \{s_1, s_2, \ldots, s_n\}$$

$$T = \{t_1, t_2, \ldots, t_m\}$$

$$\phi : T \to S$$

Where S is the set of shards, T is the set of transactions, and ϕ is the distribution function.

14 Conclusion

In this paper, we have introduced Hyra Network, a breakthrough Layer-3 blockchain architecture designed to build a decentralized, scalable, and verifiable AI compute infrastructure. By combining the power of ZK-STARKs, blockchain, and AI, Hyra Network addresses the core challenges in current AI infrastructure: accessibility, transparency, and verifiability.

We have developed a comprehensive mathematical framework for ZKML, low-latency task orchestration, fair incentive systems, and an energy-efficient PoUW consensus mechanism. Our theoretical analysis and simulations demonstrate that Hyra Network can scale to support millions of concurrent users, providing the world's largest decentralized AI computational power.

By creating a global marketplace for AI computation, Hyra Network not only improves the economic efficiency of AI infrastructure but also democratizes access to advanced AI technologies. This has the potential to usher in a new era for AI, where models and applications can be developed and deployed transparently, efficiently, and verifiably.

Our contributions pave the way for future research at the intersection of blockchain, zero-knowledge proofs, and AI, with the ultimate goal of building a truly decentralized and trustworthy global AI infrastructure.

References

[1] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2014.

[2] J. Dean et al., "Large Scale Distributed Deep Networks," in *NIPS*, 2012.

[3] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *IACR Cryptology ePrint Archive*, 2018.

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

[5] A. R. Choudhury et al., "Energy Consumption of Deep Learning: A Survey," in *IEEE TKDE*, 2022.

[6] P. Wang et al., "DePIN: Decentralized Physical Infrastructure Networks," *arXiv preprint arXiv:2306.02970*, 2023.

[7] P. Goyal et al., "Filecoin: A Decentralized Storage Network," 2017.

[8] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186-208, 1989.

[9] E. Ben-Sasson et al., "Transparent Succinct Arguments with Polylogarithmic Verification," in *EUROCRYPT*, 2022.

[10] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," 2019.

[11] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zk-SNARKs with Universal and Updatable SRS," in *EUROCRYPT*, 2020.

[12] Z. Williamson, "The STARK Verifier," 2020.

[13] D. Silver et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354-359, 2017.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[15] A. Vaswani et al., "Attention is All You Need," in *NIPS*, 2017.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[17] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *CVPR*, 2017.

[18] I. Goodfellow et al., "Generative Adversarial Nets," in *NIPS*, 2014.

[19] T. Brown et al., "Language Models are Few-Shot Learners," in *NeurIPS*, 2020.

[20] M. Zaharia et al., "Apache Spark: A Unified Engine for Big Data Processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56-65, 2016.

[21] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL-HLT*, 2019.

[22] K. Bonawitz et al., "Towards Federated Learning at Scale: System Design," in *SysML*, 2019.

[23] A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," in *ICML*, 2021.

[24] N. P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, 2017.

[25] D. Ramage and D. Mazzoni, "Federated Learning: Collaborative Machine Learning without Centralized Training Data," *Google AI Blog*, 2017.

[26] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, vol. 32, no. 2, pp. 374-382, 1985.

[27] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382-401, 1982.

[28] R. Rao and S. Vrudhula, "Energy Optimal Speed Control of a Producer-Consumer Device Pair," in *ISLPED*, 2007.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[30] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," in *OSDI*, 2016.