Tensor One Whitepaper

www.tensorone.ai

Abstract

This paper presents Tensor One, a decentralized framework for the distribution, execution, and verification of artificial intelligence workloads across a globally accessible network of user-contributed GPU nodes. As AI systems grow in complexity and demand, centralized infrastructure has become a bottleneck for scalable and equitable access. This work investigates an alternative approach in which AI tasks, ranging from inference and multimodal generation to autonomous agent operations, are decomposed and allocated by a resource-aware scheduler, executed by independent nodes, and verified through smart contract–facilitated cryptographic proofs. We design and evaluate a protocol architecture that includes resource allocation logic, escrow-based compensation, and irreversible task verification. In addition to the protocol itself, we demonstrate its applicability through a decentralized application offering visual agent creation, AI tool APIs, and model training workflows. This research contributes a novel framework for trustless, permissionless AI compute coordination and sets the foundation for a scalable, incentive-aligned machine intelligence network.

1 Introduction

Over the last decade, artificial intelligence has become a foundational layer across industries, driven by advances in generative models, agent-based systems, and increasingly large-scale inference workloads. However, the infrastructure supporting this growth remains heavily centralized, creating barriers to access, scalability, and cost-efficiency. As model sizes grow and demand accelerates, the reliance on concentrated data centers has led to bottlenecks in availability, limitations in distribution, and increased exposure to censorship and single points of failure[1].

At the same time, hundreds of millions of GPUs remain underutilized globally, sitting idle in personal computers, gaming rigs, or small-scale clusters[2]. Tensor One seeks to address this imbalance by proposing a decentralized framework that transforms idle GPU resources into an economically incentivized, verifiable compute layer for artificial intelligence. The system enables users to become nodes within a distributed network, contributing compute toward generative workloads, model training, and autonomous agent execution while earning rewards proportional to their participation.

This paper presents the architecture and implementation of the Tensor One protocol. We detail a three-stage system for task distribution and verification, comprising (1) Tensor Resource Allocation, which schedules tasks based on computational complexity and node performance; (2) Tensor Escrow, which secures compensation through smart contract–mediated reward locking; and (3) Task Verification, which ensures result validity and task irreversibility using encrypted on-chain proofs. In addition to the core protocol, we describe the Tensor Playground, a dApp offering AI-powered tools such as uncensored chat, code generation, video synthesis, and a visual agent builder.

We also introduce TrainOps, an AI model refinement system built atop the same distributed GPU infrastructure, and outline our approach for launching monetizable autonomous agents through a no-code framework. By combining cryptoeconomic incentives, verifiable computation, and decentralized infrastructure, Tensor One enables a permissionless network for scalable AI access and contributes toward a broader shift from centralized machine intelligence to globally distributed, user-powered systems.

2 Objectives

Tensor One targets three systemic bottlenecks in the development of decentralized AI infrastructure:

1. Decentralized Compute Distribution: Reduce dependency on centralized cloud platforms by enabling compute tasks to run on a globally distributed GPU mesh via browsers.
2. Formal Agent Execution Models: Utilize state machine and DAG-based runtimes to enforce determinism, reusability, and structured coordination across agent workflows.
3. Economic Enforcement via Protocol Incentives: Create mechanisms for task validation, collateralized execution, slashing penalties, and compute-based rewards that are enforced through smart contracts.

3 Task Distribution Model

Let

$$G = (V, E)$$

be a directed acyclic graph (DAG), where subtasks are their dependencies are defined as vertices and edges, respectively [1][2]:

$$V = \{V_1, V_2, \dots, V_n\}$$

is a set of subtasks (vertices), and

$$E \subseteq V \times V$$

is the set of directed edges such that

$$(V_i, V_j) \in E \Rightarrow V_j \text{ depends on } V_i$$

This model allows task execution to be organized as a dependency-aware graph, where each node represents an atomic unit of work (a subtask), and edges encode prerequisite relationships. By structuring execution in this way, we ensure that no task is processed before all of its dependencies have been satisfied that is a fundamental requirement for deterministic, parallel, or distributed compute environments. The acyclic nature of the graph guarantees that there are no circular dependencies, which simplifies scheduling and enables safe parallelism.

Within the context of Tensor One, this DAG structure is essential for distributing compute workloads across decentralized nodes. Each subtask can be assigned independently once its parent tasks are completed, allowing the system to dynamically allocate compute resources based on task readiness. This forms the basis for agent-level scheduling, progressive task execution, and efficient orchestration of large-scale AI workflows across the network.

4 Task Allocation Function

Let

$$N = N^1, N^2, \ldots, N_{\mathrm{m}}$$

be the set of available compute nodes, each equipped with GPU resources.

Let

$$Power(N_{\mathrm{j}}) = FLOPs/sec\ of\ nodeN_{\mathrm{j}}$$

Let

$$Reward(V_{\mathrm{i}}) = incentive\ value\ of\ subtask\ V_{\mathrm{i}}$$

Let

$$Deadline(V_{\mathrm{i}}) = time\ constraint\ for\ completing\ subtask\ V_{\mathrm{i}}$$

Define a utility score:

$$U_{\mathrm{ij}} = \alpha \times \big(Reward(V_{\mathrm{i}})/Deadline(V_{\mathrm{i}})\big) + \beta \times Power(N_{\mathrm{j}})$$

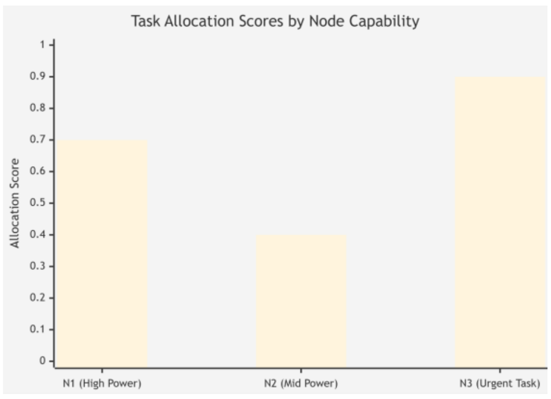
The allocation function selects the node maximizing the utility:

$$Allocate(V_{\mathrm{i}}, N) = argmax_{\mathrm{nj}} \in N(U_{\mathrm{ij}})$$

Where:

$$\alpha, \beta \in \mathbb{R}^+ (tunable\ priority\ weights)$$

Our allocation strategy is inspired by multi-criteria scheduling systems that prioritize tasks based on urgency, reward, and compute availability [1][3].



This equation decides which GPU node (N_j) gets assigned a subtask (V_i). It balances two priorities:

1. Reward vs. Urgency

$$\alpha \cdot \frac{Reward\,(V_i)}{Deadline\,(V_i)}$$

This term prioritizes tasks that are either high-value or time-sensitive.

Example: A task offering a 10 TPO reward with a 10-minute deadline will score higher than one offering 5 TPO with a 1-hour deadline.

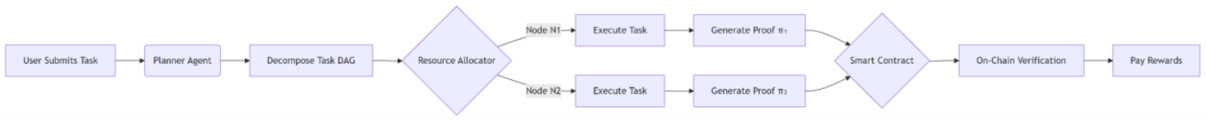
2. Node Capability

$$\beta \cdot Power\left(N_j\right)$$

This term favors allocation to higher-performance nodes (measured in FLOPs).

Example: A node with an RTX 4090 is preferred for training large language models over a mid-tier GPU.

Together, these components ensure that urgent, high-value tasks are completed efficiently by capable nodes, aligning platform incentives with compute efficiency.



Why It Matters:
Ensures efficient resource use while incentivizing nodes to compete for high-value work.

4.1 Escrow & Compensation

To ensure trustless and verifiable execution, all task rewards are initially held in escrow. These rewards are only released once the assigned node submits a valid proof of execution.

Let R be the reward for a subtask V_i, and let π_i be the proof of correct execution submitted by the compute node.

The escrow mechanism is defined as

$$Escrow(V_i, R) = SCr(R, \pi_i) \text{ } where \text{ } Verify(\pi_i) = True$$

All rewards are locked in smart contract-based escrow until cryptographic proofs of task completion are verified, ensuring integrity and trust [4][5].

4.2 Task Verification

To ensure correctness without compromising privacy, nodes submit zero-knowledge proofs π_i upon completing a subtask V_i. These proofs validate execution without revealing the underlying input or output data.

The verification function is defined as:

$$Verify(V_i, \pi_i) = Decrypt(\pi_i, Key_o) \wedge Hash(\pi_i) = Commitment(V_i)$$

Where:

- π_i is the zero-knowledge proof submitted by the node

- Key_o is the public key of the task owner
- Commitment(V_i) is the pre-registered on-chain hash of the expected proof

Zero-knowledge proofs (e.g., zk-SNARKs) are used to verify task correctness without revealing underlying data, leveraging established cryptographic protocols [4][5][8].

Proof Requirements:

- Proof π_i:
  - A cryptographic proof (e.g., zk-SNARK) that confirms the task was computed correctly.
- Commitment Match:
  - The hash of the submitted proof must match the expected commitment stored on-chain for that task.

Why It Matters:
This verification model prevents manipulation or falsified task completion. It acts as a tamper-proof receipt that guarantees honest compute without exposing sensitive data which is essential for securing trust in decentralized execution environments.

5 Incentive Mechanism

To ensure only trusted nodes are rewarded, Tensor One uses a consensus-based scoring model. A node must be vouched for by a majority of staked participants to qualify for rewards.

The reputation and consensus scoring system draws from cryptoeconomic designs similar to those in Ethereum's staking and slashing mechanisms [7][8].

5.1 Anti-Collusion

The consensus score is defined as:

$$C(N_j) = \sigma_\rho\left(\frac{1}{k}\sum_k \text{Stake}(N_k)\cdot T(N_k \to N_j) - \kappa\right)$$

Determines if a node (N_j) is trusted by the majority of the network

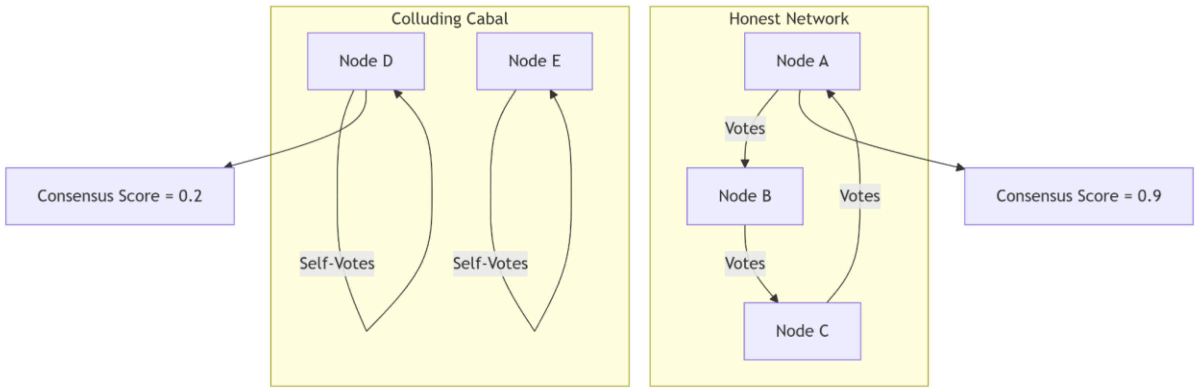
Where:

- $C(N_j)$: Consensus score for node N_j
- $T(N_k \to N_j)$: Binary trust signal (1 if node $N_k$vouches for N_j, 0 otherwise)
- Stake(N_k): The staked amount of node N_k, representing voting weight
- κ: Trust threshold (e.g., κ = 0.5 for 50% majority)
- σ_ρ: Sigmoid activation function, smoothly scaling output between 0 and 1

Trust Requirements:

- Nodes must be vouched for by more than 50% of stake-weighted peers.

- The higher the stake behind trust signals, the higher the score.
- Rewards are only issued if $C(N_j) \geq 0.5$, ensuring majority approval



Why It Matters:
This trust-based incentive model ensures that compute rewards are only distributed to nodes that are vetted by their peers. It prevents Sybil attacks, discourages collusion, and promotes reputation-based trust in an open network, all without relying on central authorities.

5.2 Bonded Rewards

Nodes accumulate **bonds** B in peers they rank highly. This mechanism allows nodes to signal trust and share in the rewards of high-performing participants.

The update rule is:

$$B_{(t+1)} = B_t + W \cdot S, \Delta S = \tau \cdot (0.5 \cdot B \top I + 0.5 \cdot I)$$

Where:

- W: Weight matrix representing peer rankings
- S: Stake vector
- $B^T I$ Bond ownership (the rewards a node receives based on bonds it holds)
- I: Identity vector representing self-reward
- τ: Reward emission rate

Incentive Breakdown:

- Bond Ownership ($B^T I$)
  - Nodes earn a portion of the rewards generated by peers they've bonded to.
  - *Example: If Alice bonds to Bob (a top performer), she receives a share of Bob's rewards*
- Self Incentive ($0.5I$)
  - 50% of rewards are kept by the node that completes the work directly.

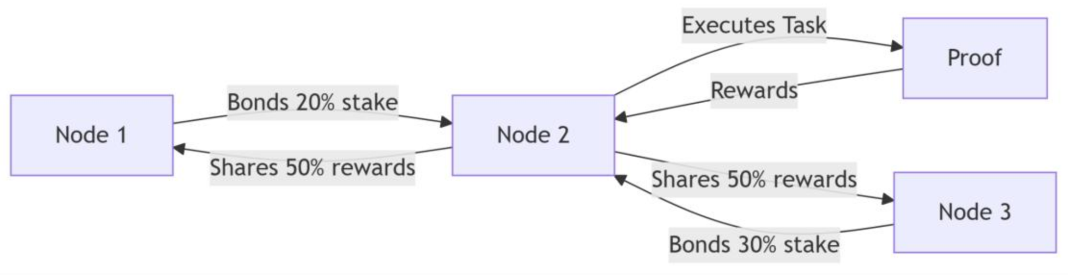


Figure 4: Bonded Reward Sharing Among Nodes

6 Workflow

The Tensor One execution pipeline follows a structured lifecycle to ensure decentralized, verifiable compute across the network:

- Task Decomposition
  - A user submits a high-level task T. his task is parsed by a Planner Agent, which decomposes it into a Directed Acyclic Graph (DAG) $G = (V, E)$, where each vertex $Vi \in V$ represents a subtask, and edges $(Vi, Vj) \in E$ encode execution dependencies (i.e., Vj cannot begin until V_i completes.
- Execution
  - Each subtask V_i s assigned to an available Executor Node based on the platform's allocation algorithm. The node performs the computation and submits a corresponding zero-knowledge proof π_i , demonstrating the validity of the result without revealing sensitive data.
- Finalization
  - A Finalizer Agent collects submitted proof $\pi_i,$ verifies them using a pre-agreed validation protocol, and upon successful verification, triggers the release of escrowed rewards. This ensures both task correctness and fair compensation for compute providers.

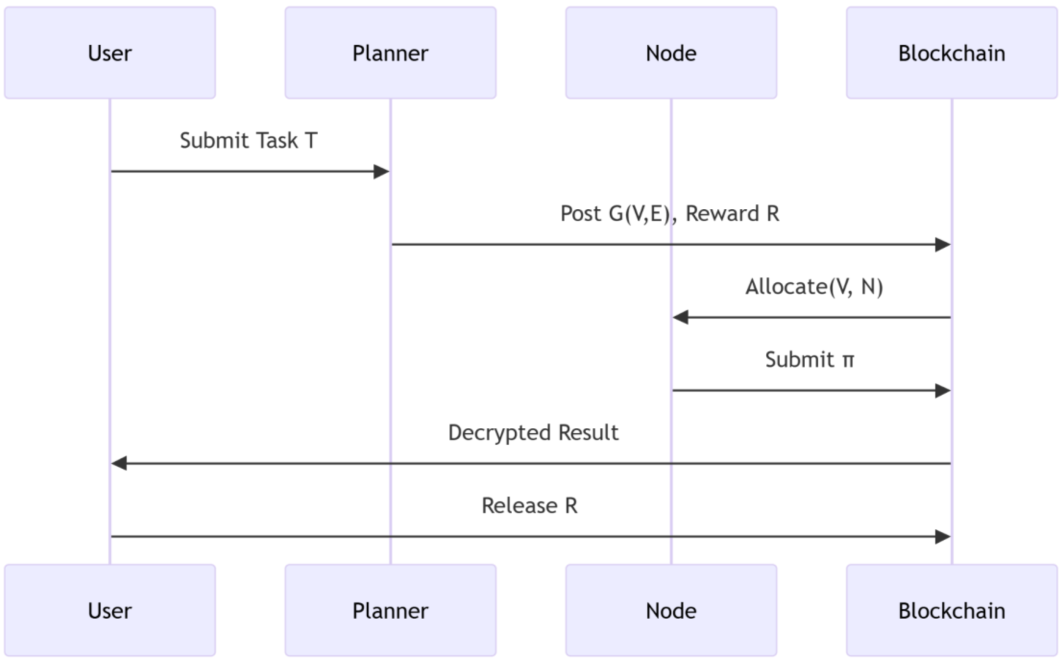


Figure 5: End-to-End Task Execution Flow in Tensor One Protocol

7 Agent Execution Layer

Autonomous agents are a powerful paradigm in decentralized AI compute as they are capable of navigating dynamic environments and executing complex objectives without centralized oversight. By leveraging Tensor One's verifiable execution layer and built-in incentive mechanisms, these agents can operate trustlessly, adaptively, and at scale.

We define an autonomous agent as a finite-state machine:

Where:

- S: Set of agent states (e.g., *idle*, *bidding*, *executing*)
- Σ: Set of possible inputs
- δ: State transition function
- s_0: Initial state
- F: Final or terminal states

The transition function is defined as:

$$\delta(st, xt) = s_{(t+1)}$$

This function determines how the agent moves between states based on input x_t at time step t.

Explanation:

- Input-Driven Transitions: The x_t (e.g., sensor data or user requests) triggers a state change. For example, detecting "traffic jam" could shift an agent from planning to re-routing.
- Deterministic Behaviour: Given the same state and input, the resulting state is always the same. This ensures predictable and verifiable execution logic.

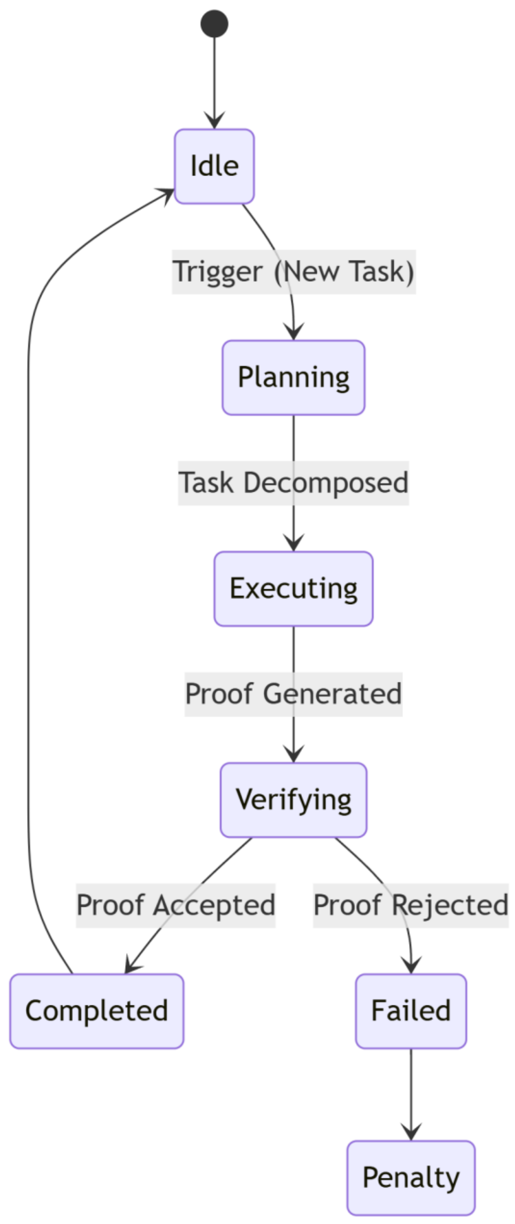


Figure 6: Agent Lifecycle State Diagram

References

[1] Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3), 260–274. https://doi.org/10.1109/71.993206

[2] Kwok, Y.-K., & Ahmad, I. (1999). Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Computing Surveys*, 31(4), 406–471. https://doi.org/10.1145/344588.344618

[3] Abadi, M., Barham, P., et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. *OSDI '16*.
https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[4**]** Ben-Sasson, E., Chiesa, A., Tromer, E., & Virza, M. (2013). Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. *USENIX Security Symposium*.
https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/ben-sasson

[5] Feige, U., Fiat, A., & Shamir, A. (1988). Zero Knowledge Proofs of Identity. *Journal of Cryptology*, 1, 77–94.