Impulse AI: Verifiable Decentralized Infrastructure for AI Training

A Technical Whitepaper.
Version 1.0

The Impulse AI Team

1 Introduction

Over the past few years, fine-tuning large pre-trained language and vision models has become the de facto way for organizations to extract bespoke capabilities from foundation models. Whether adopting a multilingual transformer for legal parlance, teaching a vision model to identify manufacturing defects, or customizing a dialog agent for customer service, fine-tuning domain-specific data reliably boosts accuracy and relevance. In practice, teams spin up GPU clusters on major cloud platforms (e.g., AWS, GCP, Azure), rent dedicated instances from specialized providers, or maintain on-premises hardware orchestrated via Kubernetes or other job-scheduling frameworks. Each training job is configured with hyperparameters, training data paths, and checkpointing policies. Engineering teams (or management) then monitor logs, scale resources up or down, and pay per GPU hour, plus storage and networking overhead.

This model, which is inherently centralized, is effective, but it has four interrelated drawbacks. First, cost and utilization inefficiencies abound: long-running clusters incur idle-time expenses, spot instances can be interrupted unpredictably, and cloud providers' opaque pricing makes it hard to compare alternatives. Furthermore, users are often subjected to a tradeoff between high on-demand pricing and inefficiency for long-term rentals. Second, trust in compute integrity is assumed rather than proven. Tricks like truncated batches (run on a third-party infrastructure) or stale gradients can reduce a model's proper training. Third, a single point of failure and a limited set of vendors constrain availability, geographic diversity, and resilience against outages or regulatory disruptions. Fourth, building a comprehensive, tailor-made machine learning infrastructure is non-trivial, often requiring labor and expertise in high demand but in short supply.

At the same time, the rapid proliferation of open-source compute providers, edge-data centers, and GPU-sharing marketplaces suggests that a more flexible, crowd-sourced approach is possible—if only there were a way to guarantee that distributed providers perform the work they claim. Traditional "spot-market" systems lack strong cryptographic assurances or economic deterrents against fraud. Smart contract systems can automate payments, but without verifiable proof of computation, they merely shift trust to oracles or reputation

systems that remain vulnerable to collusion and manipulation.

Impulse addresses these gaps by being *the* one-stop AI marketplace that is supported by decentralized infrastructure, by re-imagining fine-tuning as a fully decentralized, incentive-compatible marketplace with built-in verifiable computing. Rather than negotiating VM instances with a handful of hyperscalers, data scientists, engineers, and users in general submit fine-tuning jobs to the Impulse protocol, specifying their budget, performance objectives, and data-access controls in a smart contract. The protocol then utilizes its Orchestration Layer to match jobs with collateral-backed Compute Providers across a global network, leveraging internal scheduling heuristics and external brokering partners. Once a provider accepts a task, the system enforces verifiable training: only a fraction of gradient updates are redundantly checked by randomly selected verifiers, all of whom stake tokens they stand to lose if they rubber-stamp false results. Detected errors trigger the slashing of malicious actors, while honest participants earn proportional rewards in the native token. This user flow is exemplified in Figure 1.

2 Stakeholder Mapping

Before delving into the mechanics of the protocol, it is essential to understand who will participate, what challenges they face today, and how Impulse AI uniquely serves each group. Our discussion is summarized in Table 1 below.

Compute Providers (CPs) Many operators, from hyperscale centers down to small clusters, sit on underutilized GPUs, paying power and maintenance costs without predictable revenue. Impulse AI addresses this by matching compute providers with model trainers. In our setting, CPs can provide computational power either as provers (i.e., the actor that computes the main job) or as verifiers. Before joining Impulse, a provider stakes tokens; during execution, a random subset of steps is recomputed by verifiers. Upon successful checks, the provider receives an itemized payout, and in the event of any evidence of malicious behavior, their stake is slashed.

Model Trainers and Fine-tuners Data scientists and ML engineers today rely on powerful clouds—AWS, GCP, Azure—that can spin up custom VM types, autoscale on demand, and plug into rich data ecosystems. However, these platforms impose four interlinked tradeoffs (cf Introduction).

Impulse AI addresses these issues: users submit a job specification (dataset, model, hyperparameters) via a unified SDK or API; our orchestration layer then routes that job across clouds, edge sites, or prosumer hardware; and every compute step is cryptographically verifiable, so trainers need never wonder whether paid-for work ran. By decoupling compute from any single provider, standardizing smart-contract calls, and staking correctness, Impulse delivers flexible, cost-effective, and trustless fine-tuning.

Retail Contributors Hobbyists and small labs with one or two GPUs often find existing marketplaces inaccessible: minimum commitments are too large, payments too infrequent,

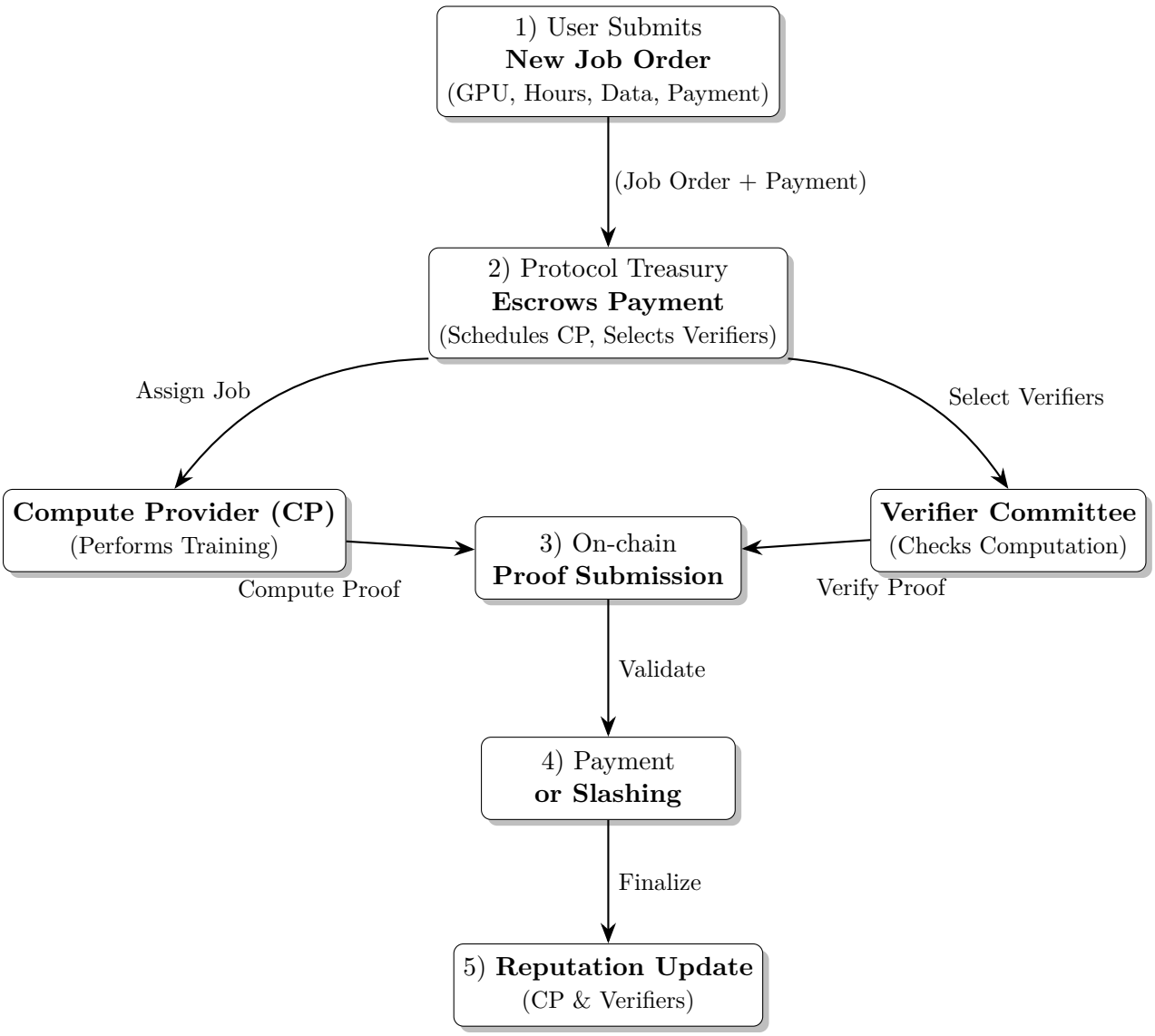

Figure 1: High-level overview of *Impulse*

and tools too complex. Impulse lowers these barriers by allowing micro-stakes and micro-transactions in native tokens. Anyone with spare GPU cycles can opt in, stake a small amount to signal their commitment and earn proportional rewards for verifiable FLOPs. Over time, retail contributors build on-chain reputations, unlocking larger tasks and higher yields. This democratizes computing, expands total network capacity, and embeds fresh geographic and hardware diversity into the protocol.

Token Holders Native token holders are custodians of the network's long-term success. Impulse AI aligns its incentives by granting governance rights over critical parameters—issuance schedules, slashing thresholds, and KPI targets—and by linking token utility to real-world adoption metrics (compute volume, fee revenue, and reliability). As the protocol scales and usage grows, strong governance participation ensures a healthy network. This creates a flywheel that rewards holders who contribute their voice and expertise to the ecosystem's evolution.

Ecosystem Partners Open-source maintainers, academic researchers, commercial integrators, and independent tool builders each bring unique value to a decentralized compute stack—but today, they navigate siloed services and one-off revenue agreements. Impulse AI provides a standardized SDK and smart-contract interfaces, enabling seamless interoperability between plugin authors, bridge developers, domain-specific orchestration adapters, and the core protocol. High-impact contributions are recognized through Retroactive Public Goods Funding rounds, on-chain bounties, and hackathons, ensuring that maintainers receive token rewards proportional to actual usage, value generated, and community endorsement. By fostering a truly community-driven development model, Impulse transforms partners into co-creators whose work directly shapes the protocol's security, performance, and longevity.

3 On Verifiable Compute

In this Section, we present a simple, mathematically rigorous proof-of-concept (PoC) mechanism for verifiable training of machine learning models in a decentralized environment. Our construction combines partial redundancy (i.e., verifying only a fraction of training steps) with economic incentives (stakes and slashing) to deter malicious behavior. In an upcoming paper, we will describe the work presented herein in more detail. We first present our methodology for a single-prover case (i.e., where a single provider performs computation) and then generalize it to multiple providers. We begin by summarizing the notation used in this section in Table 2. We summarize the state of the art of verifiable computing in the Appendix.

3.1 System Model

We consider a decentralized training framework composed of the following entities and assumptions:

1. **Prover** P (or primary worker) who performs a sequence of $\ell \in \mathbb{N}_+$ gradient-based

Stakeholder	Pain Points	How Impulse AI Helps	Token Benefits
Compute Providers	Idle accelerators, unpredictable revenue, difficult customer acquisition	Stake-backed proof-of-compute, random verifications, per-task payouts, slashing for failures	Transparent rewards, stake-driven governance weight
Model Trainers & Fine-tuners	High costs, having to build ML infra, vendor lock-in, policy constraints	Unified SDK/API, marketplace routing across clouds/edge, cryptographic proof of work	Budget-locked payments, no idle charges, governance votes on pricing & policies
Retail Contributors	High entry barrier, minimal micro-rewards, complex tooling	Micro-stakes, micro-transactions, incremental reputation, verified FLOP accounting	Earn native tokens for small jobs, reputation unlocks larger tasks
Token Holders	Need for sustainable network growth, influence over protocol direction	Governance rights over issuance, slashing, KPI targets; issuance tied to real usage metrics	Voting power, influence on parameter adjustments, alignment with protocol health
Ecosystem Partners	Siloed integrations, ad-hoc revenue, limited funding for public-goods contributions	Standard SDK/contract interfaces, on-chain bounties, RetroPGF grants, hackathons for tool creators	Token grants for contributions, community-weighted funding, seamless liquidity bridges

Table 1: Stakeholder Analysis

updates on a model parameter vector $\theta \in \mathbb{R}^d$ over a dataset partition or full dataset. Each update incurs a computational cost $C > 0$.

2. A pool of **Verifiers** $V = \{V_j\}_{j=1}^{M}$ with cardinality $M \in \mathbb{N}_+$. Each verifier can recompute a single update step for $C > 0$.

3. **Communication model**: Computation proceeds in *synchronous rounds*. The Prover broadcasts its new parameter checkpoint $(\theta_t, \mathcal{H}_t)$ to the network in each round. We assume authenticated, reliable channels (no message forgery or loss).

4. **Staking mechanism**: Prover and Verifiers deposit stakes s_P and s_V respectively, on a smart contract. Honest behavior is enforced via slashing: incorrect submissions lose their Stake.

Symbol	Definition
ℓ	Total Number of gradient-update steps in one epoch
M	Total Number of possible Verifiers
m	Size of the verifier subcommittee per checked step
α	Fraction of steps selected for redundancy verification
$\tilde{n} = \alpha\ell$	Expected number of steps sampled for verification
$\tilde{N} \subseteq \{1, \ldots, \ell\}$	Randomly chosen set of step indices to verify, $\lvert\tilde{N}\rvert = \tilde{n}$
C	Computation cost per update for the Prover
C_v	Computation cost per update for each Verifier
θ_t	Model parameter vector after step t
$\mathcal{H}_t$	Auxiliary state at step t (e.g. random seed, hyperparameters)
$\mathcal{L}_t$	Training data (or its encrypted form) at step t
f	Number of steps on which the Prover attempts to cheat
$\delta(f)$	Probability of detecting at least one incorrect update among f cheats
G	Expected gain from cheating on a single step
s_P	Stake deposited by the Prover
$p_{\text{undetected}}$	Probability a fraudulent step passes verification
$p_{\text{detected}} = 1 - p_{\text{undetected}}$	Probability a fraudulent step is caught
N_m	Number of workers in distributed (DiLoCo) setting
k	Number of local gradient steps per outer iteration
$\ell_{\text{out}} = \ell/k$	Number of outer synchronization rounds
α_{out}	Fraction of outer rounds sampled for verification
m_{out}	Committee size for outer-round verification
$\tilde{n}_{\text{out}}$	Number of outer rounds sampled: $\alpha_{\text{out}}\ell_{\text{out}}$
τ_{out}	Tolerance threshold for norm-difference in block updates

Table 2: Notation

3.2 Threat Model

We allow for the following adversarial capabilities:

1. **Prover corruption**: The Prover may deviate by submitting incorrect updates on up to $f_P < \ell$ distinct steps.
2. **Verifier collusion**: Up to $f_V < m$ verifiers in any subcommittee of size m may collude to approve a malicious update. We require $m > f_V$.
3. **Network adversary**: Cannot forge or block messages but may delay messages by up to one round.

Under these assumptions, the protocol must satisfy:

- **Soundness**: Any incorrect update is detected with high probability.
- **Liveness**: Honest Prover and Verifiers can complete all ℓ steps in $O(\ell)$ rounds.
- **Incentive Compatibility**: Rational participants maximize expected utility by be-

having honestly.

3.3 Single-Prover Verifiable Compute

We follow an approach similar to that of [22], albeit with the goal of improving its efficiency through crypto-economic systems. We consider a setting with one primary *Prover*, who trains a model by performing $\ell \in \mathbb{N}_+$ gradient-based updates in a single epoch. Each update costs $C \in \mathbb{R}_+$ for the Prover. We assume there are $M \in \mathbb{N}_+$ *Verifiers*, each capable of recomputing a single training step at cost $C_v \in \mathbb{R}_+$. In a naive scheme, all $M \in \mathbb{N}_+$ Verifiers would recompute *every* step, incurring total verification cost on the order of $M\ell C_v$. Our goal is to reduce this overhead without compromising verifiability.

To that end, only a fraction $\alpha \in (0, 1]$ of the training steps are verified in our approach. Let $\tilde{n} := \alpha\ell$ denote the expected number of steps selected for verification, where $\tilde{n}$ is chosen randomly or by some protocol-specific rule. For each selected step, only a subcommittee of size $m \ll M$ is tasked with recomputing the update. This subcommittee then confirms or refutes the correctness of the Prover's submission. The main idea is illustrated in Algorithm 1.

Algorithm 1: Single-Prover Partial Redundancy Verification

Protocol determines the set of steps to check $\tilde{N}$ to check, $|\tilde{N}| = \tilde{n}$. These steps are kept secret from both provers and verifiers, to avoid potential collusions.
for $t = 1 \dots \ell$ **do**
  Prover does one model update $\theta_{t+1} = \mathsf{Update}(\theta_t, \mathcal{L}_t, \mathcal{H}_t)$, with $\mathcal{L}_t, \mathcal{H}_t$ the model data and hyper-parameters respectively.
  Prover encrypts $\mathcal{L}_{t+1}$
  if $t \in \tilde{N}$ **then**
    Prover saves θ_t (saves computational costs)
    Protocol choose m out of M possible validators
    for $j = 1 \dots, m$ **do**
      Compute $\theta^j_{t+1} = \mathsf{Update}(\theta_t, \mathcal{L}^j_t, \mathcal{H}_t)$ and L^j_{t+1}
    end for
    Determine anomalies in the distances within validators and between validators and the Prover
  end if
end for

In a committee-based[1] naive "fully redundant" PoL approach (cf. [22]), each of the M Verifiers checks every one of the ℓ steps. The system's *verification cost* then becomes:

$$\mathrm{Cost}^{\mathrm{naive}}_{\mathrm{verify}} = M\,\ell\,C_v$$

Adding in the Prover's training cost $\ell\,C$ yields an overall system cost of:

$$\mathrm{Cost}^{\mathrm{naive}}_{\mathrm{total}} = \ell\,C + M\,\ell\,C_v$$

[1]We remark that while [22] uses a single *verifier* (e.g., a committee with $M = 1$ members), this is prone to malicious attacks (e.g., collusion) so we resort to a committee approach with multiple members.

In contrast, our partial redundancy protocol requires only $\tilde{n} = \alpha\ell$ steps to be rechecked, each by m Verifiers. The verification cost becomes:

$$\text{Cost}_{\text{verify}} = \tilde{n}\, m\, C_v = \alpha\, \ell\, m\, C_v$$

Thus, the total system cost is:

$$\text{Cost}_{\text{total}} = \ell\, C + \alpha\, \ell\, m\, C_v$$

We achieve a strictly lower verification overhead if:

$$\alpha\, m \ll M, \quad \text{i.e.} \quad \alpha\, m\, C_v < M\, C_v + \mathsf{BandwidthCosts}$$

Hence, the partial redundancy mechanism can reduce the total cost by $\frac{\alpha\, m}{M}$ relative to naive approaches. At the same time, the stake-and-slash component enforces honest participation.

3.3.1 Detection Probability and Incentive Alignment

Let f denote the number of gradient-update steps on which the Prover attempts to cheat. We sample without replacement a set $\tilde{N} \subseteq \{1, \ldots, \ell\}$ of size $\tilde{n} = \alpha\ell$. The probability that none of the f malicious steps lies in $\tilde{N}$ is given by the hypergeometric distribution:

$$\Pr[\text{no detection}] = \frac{\binom{\ell - f}{\tilde{n}}}{\binom{\ell}{\tilde{n}}}.$$

Hence, the probability of detecting at least one bad step satisfies

$$\delta(f) \;=\; 1 \;-\; \frac{\binom{\ell - f}{\tilde{n}}}{\binom{\ell}{\tilde{n}}}.$$

For the regime $f \ll \ell$, a bound via the union estimate yields

$$\delta(f) \;\geq\; 1 - \left(1 - \frac{f}{\ell}\right)^{\tilde{n}}. \tag{1}$$

Remark 1. *Notice that Equation (1) does not depend on the committee size m; instead it only depends on $\tilde{n} = \alpha\ell$. This is due to the fact that the m parameter affects the Byzantine tolerance and robustness of the verifiers, rather than the detection probability.*

Lemma (Detection Bound). *Under uniform sampling of $\tilde{n}$ steps out of ℓ, the detection probability $\delta(f)$ satisfies the above expressions.*

Proof Sketch. *Immediate from the complement of the hypergeometric probability of drawing 0 bad items in $\tilde{n}$ draws.*

We now analyze the Prover's incentives. Let $G > 0$ denote the expected gain saved by cheating on a single step, and let $s_P > 0$ be the amount the Prover has staked (and risks losing if caught). The Prover's expected utility when cheating on f steps is

$$U_{\text{cheat}}(f) = (1 - \delta(f))\, f\, G \;-\; \delta(f)\, s_P.$$

An honest strategy yields zero utility, $U_{\text{honest}} = 0$. To ensure honesty is the strict best response, we require

$$U_{\text{cheat}}(f) < 0 \quad \Longleftrightarrow \quad (1 - \delta(f))\, f\, G \;<\; \delta(f)\, s_P,$$

for all $1 \le f \le \ell$. In particular, enforcing the single-step condition $f = 1$ suffices:

$$s_P \;>\; \frac{1 - \delta(1)}{\delta(1)}\, G.$$

Theorem (Incentive Equilibrium). *If the Prover's stake s_P satisfies $s_P > \frac{1-\delta(1)}{\delta(1)}\, G$, then honest behavior maximizes the Prover's expected utility.*

Proof Sketch. *For $f = 1$, the expected payoff becomes negative by assumption. For larger f, the detection probability $\delta(f)$ is nondecreasing in f, tightening the incentive constraint further. Thus, no profitable deviation exists.*

This analysis quantifies the interplay between the sampling rate α, committee size $m < M$ (which influences $\delta(f)$), and the required stake s_P, guaranteeing that rational Provers will adhere to honest computation.

3.4 Distributed, Verifiable Compute

We now extend the ideas above to the case of multiple workers, potentially with different hardware and locations. Our methodology is based on [13] and [12], which have shown great promise in distributed training.

Specifically, *Distributed Low-Communication* training (DiLoCo) [13] is a decentralized training strategy that significantly reduces communication overhead by allowing each worker to perform multiple local gradient steps before periodically synchronizing with other workers. Concretely, every worker $i = 1, \ldots, N_m$ maintains a local copy of the model parameters θ_t and updates it using standard mini-batch gradient methods (e.g., Adam) over a local dataset or local subset of data, accumulating k steps before communication. After k local updates, workers exchange only their updated parameters (or a compressed form) and compute a global average: $\theta_{t+1}^{\text{global}} \leftarrow \frac{1}{N} \sum_{i=1}^{N_m} \theta_{t+1}^{(i)}$. Each worker then resets its local parameters to this average, and the process repeats. By reducing the frequency and volume of parameter exchanges (from every step to every k step), DiLoCo dramatically cuts down on communication costs. At the same time, empirical evidence shows that final accuracy remains comparable to fully synchronous (per-step) methods when k is chosen suitably. This approach is robust to heterogeneous computing environments. It can accommodate nodes that intermittently join or leave the training process, making it ideal for large-scale, decentralized machine learning systems with bandwidth or latency constraints.

The idea behind enforcing verifiability on these methods is quite simple. At each *outer step*, the protocol randomly selects a subset of the providers based on some criteria, and those providers evolve the inner steps using the verification methodology outlined in Algorithm 1. Once again, the staked and rewarded amounts are chosen sufficiently large to ensure *incentive compatibility*. This is illustrated in Figure 2.

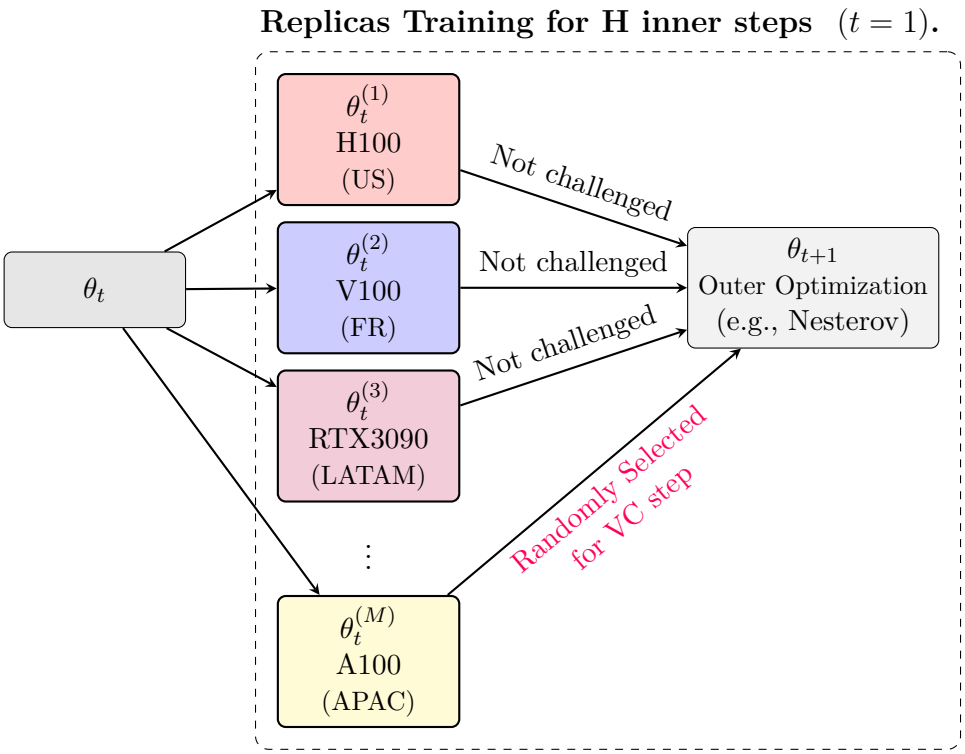

Figure 2: Verifiable DiLoCo. Figure adapted from [13]

3.5 Formalization

We extend the single-prover scheme to the DiLoCo setting of Douillard et al. (2023, 2025). Suppose there are N_m workers, each performing k local gradient steps per outer iteration, for a total of

$$\ell_{\text{out}} = \frac{\ell}{k}$$

outer rounds. At outer iteration $u = 0, \ldots, \ell_{\text{out}} - 1$, worker i starts from $\theta_{uk}^{(i)}$ and applies

$$\theta_{uk+t}^{(i)} = \mathsf{Update}\Big(\theta_{uk+t-1}^{(i)}, \mathcal{L}_{uk+t}, \mathcal{H}_{uk+t}\Big), \quad t = 1, \ldots, k,$$

Then, it computes its increment

$$\Delta\theta_u^{(i)} = \theta_{(u+1)k}^{(i)} - \theta_{uk}^{(i)}.$$

All workers broadcast $\Delta\theta_u^{(i)}$, compute the global average

$$\theta_{(u+1)k}^{\text{global}} = \frac{1}{N_m} \sum_{i=1}^{N_m} \Big(\theta_{uk}^{(i)} + \Delta\theta_u^{(i)}\Big),$$

and reset their local copy to this average.

To enforce verifiability, we sample without replacement a set $\tilde{N}_{\text{out}} \subseteq \{0, \ldots, \ell_{\text{out}} - 1\}$ of size

$$\tilde{n}_{\text{out}} = \alpha_{\text{out}}\, \ell_{\text{out}}.$$

For each sampled outer index u, a committee of size m_{out} recomputes a handful of the local updates for a selected worker i:

$$\widehat{\Delta\theta}_u^{(i,j)} = \mathsf{LocalCompute}\Big(\theta_{uk}^{(i)}, \{\mathcal{L}_{uk+t}\}_{t=1}^{k}\Big), \quad j = 1, \ldots, m_{\text{out}},$$

and checks

$$\Big\|\widehat{\Delta\theta}_u^{(i,j)} - \Delta\theta_u^{(i)}\Big\| \leq \tau_{\text{out}}.$$

Any deviation beyond τ_{out} triggers the dishonest party's stake slashes.

Algorithm 2: Verifiable DiLoCo

Input: total updates per epoch: ℓ, local steps per outer round: k, sampling fraction: α_{out}, committee size: m_{out}
Compute $\ell_{\text{out}} = \ell/k$;
Sample a set $\tilde{N}_{\text{out}} \subseteq \{0, \ldots, \ell_{\text{out}} - 1\}$ with $|\tilde{N}_{\text{out}}| = \alpha_{\text{out}} \cdot \ell_{\text{out}}$;
for $u = 0$ **to** $\ell_{\text{out}} - 1$ **do**
  Each worker i performs k local updates $\rightarrow \Delta\theta_u^{(i)}$;
  Workers broadcast $\Delta\theta_u^{(i)}$;
  Compute $\theta_{(u+1)k}^{\text{global}} \leftarrow (1/N_m) \sum_i (\theta_{uk}^{(i)} + \Delta\theta_u^{(i)})$;
  if $u \in \tilde{N}_{\text{out}}$ **then**
    Form a verifier committee of size m_{out};
    Each verifier recomputes the block update and checks $\|\widehat{\Delta\theta}_u^{(i,j)} - \Delta\theta_u^{(i)}\| \leq \tau_{\text{out}}$;
    Slash any party with deviation $> \tau_{\text{out}}$;
  end if
end for

In this Section, we have presented a unified framework for verifiable computing that rigorously addresses both single-prover and fully distributed training scenarios. By combining partial-redundancy sampling (with hypergeometric detection bounds) and economic incentives (stake-and-slash), we guarantee soundness, liveness, and incentive-compatibility even in the presence of malicious or colluding parties. Our extension to the DiLoCo setting preserves the low-communication benefits of decentralized training while embedding verification at the "outer" synchronization level—ensuring correctness across heterogeneous nodes and geographies.

In the following section, we will describe our orchestration layer, i.e., how we assign jobs to CPs.

4 Orchestration Layer (Job Scheduling)

Our product architecture features a scheduling mechanism that provides users with the best compute provider. We now describe how this is done.

4.1 Problem Setup

We consider a decentralized job scheduling framework involving a set of workers, $\mathcal{W} = \{1, 2, \ldots, M\}$, and a set of jobs, $\mathcal{J} = \{1, 2, \ldots, N\}$. Each job $j \in \mathcal{J}$ is characterized by a budget $B_j \in \mathbb{R}^+$ and a load $L_j \in \mathbb{R}^+$, while each worker $i \in \mathcal{W}$ is defined by a capacity $K_i \in \mathbb{R}^+$, a quality score $\mathsf{Rep}_i \in [0, 1]$, and a bid $p_{i,j}$ that represents their willingness to execute job j.

The objective is to determine a binary assignment matrix $X = [x_{i,j}]$ with $x_{i,j} \in \{0, 1\}$ that maximizes the overall utility given by:

$$U_{\text{protocol}}(X) = \sum_{i=1}^{M} \sum_{j=1}^{N} \Big(\beta\Big(B_j - p_{i,j}\Big) - \alpha(1 - \mathsf{Rep}_i)^2\Big)\, x_{i,j}$$

where $\beta > 0$ scales the economic benefit and $\alpha \geq 0$ imposes a penalty for assigning jobs to lower-quality workers. This optimization is subject to the constraints:

$$\sum_{j=1}^{N} L_j x_{i,j} \leq K_i \quad \forall\, i \in \mathcal{W}$$

$$\sum_{i=1}^{M} x_{i,j} \leq 1 \quad \forall\, j \in \mathcal{J}$$

It can be shown that this problem is NP-hard via a reduction from the *Multiple Knapsack Problem*, indicating that finding a globally optimal solution is computationally intractable for large instances.

To simplify this, we consider a uniform pricing model where $p_{i,j} = p = \beta' B_j$ for all i and j. Under this assumption, the utility function reduces to:

$$U_{\text{protocol}}(X) = \sum_{i=1}^{M} \sum_{j=1}^{N} \Big(rB_j - \alpha(1 - \mathsf{Rep}_i)^2\Big)\, x_{i,j}, \quad \text{with } r = \beta - \beta'$$

This problem is also NP-hard and as such we propose greedy approximation algorithms. This can be done with a *naive* greedy algorithm, where jobs are sorted according to their proposed utility and chosen until the total capacity is reached (with complexity $O(MN)$) or via a *Capacity-Degrading Greedy Algorithm* (**CDA**), each worker's effective score is dynamically adjusted as capacity is consumed:

$$\text{EffScore}_i = \mathsf{Rep}_i \times \left(\frac{K_i}{\text{OrigCap}_i}\right)^{\gamma}$$

Here, $\gamma > 0$ controls the degradation rate. For each job, a binary search identifies the first worker with sufficient remaining capacity, and a range-max query (using a balanced data structure such as a segment tree) selects the worker with the highest effective score. Our internal research shows that this method runs in $O(M \log M + N \log N + N \log M)$ time and requires $O(M + N)$ space.

Empirical results, obtained from synthetic datasets, indicate that both algorithms can handle large-scale instances efficiently (even for $M, N \approx 10,000$), with the **CDA** method generally achieving higher overall utility by better balancing job value and worker quality. Results are shown in Figure 3.

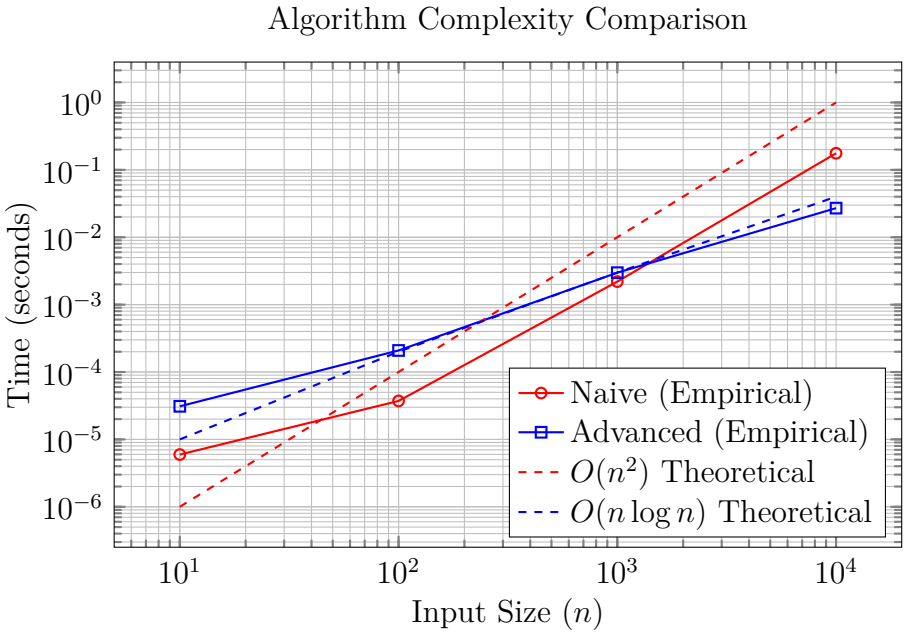

Figure 3: Complexity of CDA vs Greedy

4.1.1 Example

To illustrate our scheduling in action, consider a minimal network with two workers, A and B, and two jobs, j_1 and j_2. Worker A has capacity $K_A = 8$ and reputation $\text{Rep}_A = 0.9$, while B has $K_B = 5$ and $\text{Rep}_B = 0.7$. Job j_1 arrives with load $L_1 = 4$ and budget $B_1 = 50$, and j_2 with $L_2 = 6$ and $B_2 = 30$. We set economic scaling $\beta = 1$ and reputation penalty coefficient $\alpha_{\text{pen}} = 2$.

Initially, effective scores are

$$\text{EffScore}_A = 0.9 \cdot \left(\tfrac{8}{8}\right)^{\gamma},$$
$$\text{EffScore}_B = 0.7 \cdot \left(\tfrac{5}{5}\right)^{\gamma}.$$

Sorting jobs by descending B_j/L_j gives j_1 before j_2. We assign j_1 to A (highest score) and reduce A's remaining capacity to 4, updating

$$\text{EffScore}_A \leftarrow 0.9 \cdot \left(\tfrac{4}{8}\right)^{\gamma}.$$

Next, j_2 (load 6) cannot fit on A (remaining capacity 4), so we assign it to B. The final assignment matches the external scheduler's choice, and execution proceeds.

The resulting protocol utility is

$$\begin{aligned} U_{\text{proto}} &= \left(50 - 2\,(0.1)^2\right) + \left(30 - 2\,(0.3)^2\right) \\ &= 49.98 + 29.82 = 79.80, \end{aligned}$$

slightly higher than the naive greedy, which ignores degradation. In any case, we remark that both of these are still *approximations* to the optimal solution. These approximate solutions might coincide with the optimal one, especially for smaller-sized problems, but this is not necessarily true, in general.

4.1.2 Practical Workflow in Depth

In a real-world deployment, the Orchestration Layer unfolds through the following phases:

1. **Job Submission**
   The user crafts a transaction payload containing data references, model hyperparameters, target metrics, and a budget, denoted as B_j. This payload is signed and broadcast to the Impulse smart contract, where the budget is escrowed.

2. **Job Allocation**
   Once the external assignment confirms which workers will serve each job, the internal scheduler finalizes any on-chain binding of $x_{i,j}$ and calculates per-worker stakes s_P. It then issues execution tickets, specifying compute tasks and requirements for each worker.

3. **Verifiable Execution & Finalization**
   Workers consume the staked funds to initiate verifiable compute (Section 3), running either single-prover or DiLoCo protocols. Upon successful proof submission, the user's budget is disbursed as payment. If any worker fails or is caught cheating, their Stake is slashed and redistributed to honest participants.

Throughout these steps, on-chain events maintain transparency, while off-chain computation ensures scalability. The interplay of staking, timelocks, and collateralization enforces honest behavior from end to end.

4.2 Future work: External Scheduling

As a natural next step, we will introduce an **External Scheduler** that extends Impulse's reach by inviting off-chain solvers to compete for job assignments. This can be thought of as *1 inch, for decentralized computing.* Each external solver q would stake collateral and submit a proposed assignment matrix X_q that maximizes the same protocol utility

$$U_{\text{proto}}(X_q) = \sum_{i,j} \left(\beta\,(B_j - p_{i,j}) - \alpha\,(1 - \mathsf{Rep}_i)^2\right) x_{i,j}^{(q)}.$$

At the close of a predetermined bidding window, Impulse's smart contract would evaluate all proposals $\{X_q\}$, select the highest-scoring X_{q^*}, and split the surplus between the

winning solver and the protocol. Any solver whose proposal violates capacity or budget constraints would forfeit part of its Stake, ensuring honesty. By transforming job routing into a decentralized "intent market," this extension will enable seamless brokering across public clouds, edge networks, and federated clusters—amplifying Impulse's internal Scheduling with a broad ecosystem of external compute resources.

The prices in the optimization problem can be understood as the GPU price for different compute providers (e.g., Aethir [50], Akash [1], Impulse, etc). In this setting, solving the optimization problem subject to the constraints has a clear meaning: *find the best possible allocation of CPs for the job at hand*. Borrowing some inspiration from *intent markets*, we can then create the following protocol:

External Allocation Protocol

1. Given a collection of Q schedulers,
2. Each scheduler q submits a solution X_q to the optimization problem subject to the constraints
3. The protocol chooses the solution X_q^* that provides the maximum utility $U(X_q^*)$
4. This utility is split between the protocol and the chosen scheduler

Thus, in this algorithm, each scheduler competes by proposing a solution that maximizes the overall utility. The protocol then selects the solution that achieves the highest utility, and the corresponding reward is split between Impulse and the winning scheduler. This mechanism enables the protocol to indirectly partner with external providers and generate revenue even when the routing occurs through another protocol, echoing the principles seen in intent markets.

> **Remark:** Notably, schedulers would need to *stake* some collateral as a guarantee that X_q is indeed a right solution.

The Orchestration Layer we present bridges user intent and provable execution with two interlocking mechanisms. External Scheduling harnesses market competition—requiring schedulers to stake collateral and compete on utility—to guarantee optimal routing across heterogeneous compute backends. Internal Scheduling then leverages capacity-aware heuristics and reputation scoring to pack jobs efficiently into Impulse's verifiable-compute fabric.

In the next sections, we build on this foundation by detailing reputation accumulation (Section 5), slashing and locking policies (Section 6), and the token economy that aligns network growth with stakeholder rewards (Sections 7–8).

5 Reputation

To allocate jobs fairly and securely in our protocol, each Compute Provider is assigned a dynamic reputation score, $\mathsf{Rep}(i, n)$, which is updated after the completion of each job n. This reputation score serves as an on-chain metric that encapsulates the provider's historical

performance and current Stake, while also incorporating penalties for missed heartbeats, non-malicious job failures, and malicious activity. The reputation score is defined as:

$$\mathsf{Rep}(i,n) = w_1 \cdot \text{SuccessRate}_{i,n} + w_2 \cdot \frac{\log(1+\text{TotalWorkValue}_{i,n})}{\sum_j \log(1+\text{TotalWorkValue}_{j,n})} + w_3 \cdot \frac{\text{StakeLevel}_{i,n}}{\sum_j \text{StakeLevel}_{j,n}}$$

where w_1, w_2, and $w_3 \in (0,1)$, $w_1 + w_2 + w_3 = 1$, are tunable weights that determine the relative importance of successful job completion, cumulative work value, and staked collateral respectively.

The *Success Rate* component measures the reliability and accuracy of a provider's work. For job n, let $r_{i,n}$ denote the performance rating, which is determined by the outcome of the job: a value of 1 indicates a fully successful job with all required heartbeat signals received; lower values indicate deficiencies due to missed heartbeats or non-malicious job failures; and if malicious behavior is detected, $r_{i,n}$ is set to a value approaching 0. The SuccessRate is updated using exponential smoothing:

$$\text{SuccessRate}_{i,n} = \begin{cases} r_{i,0} & \text{if } n = 0 \\ \alpha_1 \, r_{i,n} + (1-\alpha_1)\,\text{SuccessRate}_{i,n-1} & \text{if } n > 0 \end{cases}$$

with $\alpha_1 \in (0,1)$ ensuring that recent performance receives greater weight while still incorporating the complete history.

The *TotalWorkValue* reflects the cumulative economic value of completed jobs, measured in terms of budget or tokens earned. If $W_{i,n}$ represents the value associated with job n, then the cumulative work is similarly updated via exponential smoothing:

$$\text{TotalWorkValue}_{i,n} = \begin{cases} W_{i,0} & \text{if } n = 0 \\ \alpha_2 \, W_{i,n} + (1-\alpha_2)\,\text{TotalWorkValue}_{i,n-1} & \text{if } n > 0 \end{cases}$$

where $\alpha_2 \in (0,1)$. A logarithmic transformation is applied in the overall reputation score to mitigate the risk of spamming low-value tasks and to smooth out growth.

The *StakeLevel* component represents the amount of tokens a provider stakes when taking on a job, thus reflecting their commitment. This value is updated similarly:

$$\text{StakeLevel}_{i,n} = \begin{cases} \text{StakeLevel}_{i,0} & \text{if } n = 0 \\ \alpha_3 \,\text{StakeLevel}_{i,n} + (1-\alpha_3)\,\text{StakeLevel}_{i,n-1} & \text{if } n > 0 \end{cases}$$

with $\alpha_3 \in (0,1)$.

In addition to these baseline metrics, the reputation mechanism incorporates specific event-driven adjustments. A provider's reputation is increased when a job is completed successfully

with all heartbeat signals received. Conversely, if a provider misses a required heartbeat, the corresponding $r_{i,n}$ is penalized. Furthermore, a non-malicious job failure results in a moderate penalty, while detection of malicious behavior causes a substantial reduction in $r_{i,n}$ and triggers slashing of staked tokens.

The on-chain implementation is straightforward: each provider's reputation is maintained in a mapping (address $\rightarrow$ reputation state), and updates are performed through simple arithmetic operations. The use of exponential smoothing guarantees that the reputation score remains sensitive to recent performance without disregarding historical reliability, thereby minimizing the risk of gameability or score farming.

By incorporating value thresholds, adaptive smoothing, task-count penalties, and enforced stake lock-ups, the enhanced reputation mechanism becomes robust against farming, heartbeat gaming, and Sybil attacks. This dynamic, on-chain score not only feeds directly into the Internal Scheduler's efficiency heuristics (Section 4) but also underpins the slashing and reward distributions detailed in Sections 6 and 7. Together, these layers complete the secure, incentive-aligned fabric of Impulse's decentralized AI compute marketplace.

6 Locking and Slashing

To secure honest participation, both Provers and Verifiers must lock collateral and face slashing upon misbehavior. We denote a Prover's locked Stake by S_{Prover} and a Verifier's by S_{val}. A slashing event removes a fraction ς of this Stake—ς_{mal} for malicious fraud and smaller ς_{nm} for non-malicious faults—ensuring economic disincentives align with protocol goals.

6.1 Malicious Fraud

When a Prover is caught submitting a fraudulent gradient update, the protocol applies a slashing fraction $\varsigma_{\text{mal}} \in (0, 1]$ to S_{prover}. To guarantee incentive compatibility, we require that

$$\varsigma_{\text{mal}}\, S_{\text{prover}} \;\geq\; \frac{R}{p_{\text{detected}}}\,,$$

where R is the per-step reward and $p_{\text{detected}} = 1 - p_{\text{undetected}}$ is the probability of catching a cheat. This ensures that expected losses from slashing exceed any gain from evading compute cost C. Upon slashing, the Prover's instantaneous performance rating $r_{i,n}$ is set to zero, and the reputational update uses

$$\Phi(\varsigma_{\text{mal}}) = 1 - e^{-\tau\,\varsigma_{\text{mal}}}$$

to apply the maximum reputational penalty (Section 5.3). A slashed Prover is also temporarily removed from both external and internal scheduling pools for a ban duration T_{ban} blocks, preserving liveness by rerouting their jobs to honest participants.

6.2 Non-Malicious Faults

Hardware faults, out-of-memory events, or transient network outages should be subject to lighter penalties. Let p_{nm} be the per-step probability of such a fault, $C_{\text{nm}} = \xi\, C$ the wasted

compute cost with $\xi \in (0,1)$, and P_{nm} the slashed amount. The expected utility per step under non-malicious faults becomes

$$\mathbb{E}[\Sigma_{\text{nm}}] = (1 - p_{\text{nm}})(R - C) - p_{\text{nm}}(C_{\text{nm}} + P_{\text{nm}}).$$

Requiring $\mathbb{E}[\Sigma_{\text{nm}}] \geq 0$ yields

$$P_{\text{nm}} \;\leq\; \frac{(1 - p_{\text{nm}})(R - C)}{p_{\text{nm}}} - C_{\text{nm}},$$

and in practice we choose P_{nm} well below this upper bound so that isolated failures remain economically viable. Each non-malicious fault also reduces the Prover's success rate by a small fraction $\Delta_{\text{nm}} \ll 1$, then smooths via exponential decay (Section 5.3), differentiating genuine system unreliability from deliberate cheating.

6.3 Availability and Heartbeat Penalties

Compute Providers must maintain liveness even between jobs. We track an on-chain "unavailability counter" (U_i) for each missed heartbeat outside active proof windows. Instead of slashing tokens for being offline, we penalize them by a fixed fraction of their rewards Δ_{hb} per missed ping, and if U_i exceeds a threshold $U_{\max}$, we extend the provider's stake lock-up period by an additional ΔT_{hb} blocks. This approach distinguishes transient downtime from compute-time faults and ensures only persistent unavailability is economically discouraged. Notice that we also intend to have "remedial" messages, so that participants can be waived this penalty in case of, e.g., power outages, scheduled maintenance, etc.

6.4 Dynamic Lock-Up and Release

To prevent rapid stake-cycling (i.e., locking and unlocking), any newly staked collateral must remain locked for a minimum period ΔT_{min}. Thereafter, tokens unlock gradually according to a linear schedule over ΔT_{unlock} blocks. Crucially, both ΔT_{min} and ΔT_{unlock} are functions of the provider's current reputation $\mathsf{Rep}(i)$: higher-reputation participants enjoy shorter lock-up windows, whereas lower-reputation actors face longer durations. This design rewards consistent, honest behavior with enhanced liquidity while maintaining security.

6.5 Integration with Reputation and Scheduling

Every slashing event emits an on-chain log that triggers two downstream effects. First, the reputation contract pulls the slashing ratio

$$\varsigma_{i,n} = \frac{\Delta S_{i,n}}{S_{i,n-1}}$$

and computes the reputational hit via $\Phi(\varsigma_{i,n})$, feeding into the exponential smoothing of SuccessRate (Section 5.3). Second, the Orchestration Layer (Section 4) re-evaluates any in-flight or queued jobs: providers with slashing above $\varsigma_{\max}$ are temporarily blacklisted, and their assignments are re-allocated to maintain liveness and performance.

6.6 Parameter Illustration

A concrete configuration might set $\varsigma_{\text{mal}} = 0.75$, $P_{\text{nm}} = 0.05C$, $\Delta_{\text{nm}} = 0.02$, $\delta_{\text{hb}} = 0.01$, $\Delta T_{\min} = 10{,}000$ blocks, and $T_{\text{ban}} = 100{,}000$ blocks for malicious faults. These values can be adjusted via governance to balance robustness against usability.

By distinguishing malicious fraud, non-malicious faults, and mere unavailability—and by tying stake lock-up durations to reputation—this Locking & Slashing framework completes the incentive loop across verifiable compute (Section 3), job scheduling (Section 4), and dynamic reputation (Section 5). Economic penalties directly impact reputation scores, which in turn influence future lock-up terms and scheduling priorities, thereby forming a cohesive and resilient protocol for decentralized AI computation.

7 Rewards Mapping

We follow an approach similar to [58]. We begin by presenting a table summarizing our notation.

Symbol	Meaning
v_t	New tokens minted (inflationary issuance) at epoch t
$\Phi(t)$	Total transactional (fee) revenue in tokens at epoch t
q	Fraction of fees allocated to rewards
γ_{CP}, γ_{V}, γ_{B}	Fractions of v_t assigned to Compute Providers, Verifiers, and Block Proposers (sum to 1)
$\mathsf{Rep}(i,t)$	Reputation score of actor i at epoch t (Section 5)
$\mathsf{Contribution}(i,t)$	Normalized proof-of-compute volume of i in epoch t (Section 7.3)
$\mathsf{Stake}(i,t)$	Normalized stake of i at epoch t (Sections 5 & 6)
$R_{\mathsf{infl}}(i,t)$	Actor i's share of inflationary rewards at epoch t
$R_{\mathsf{fees}}(i,t)$	Actor i's share of fee-driven rewards at epoch t

Every Compute Provider (CP) and Verifier i earns rewards at epoch t according to a combination of inflationary issuance and fee-driven pools. We first partition inflationary issuance, then allocate both sources proportionally based on reputation, contribution, and stake.

7.1 Inflationary Issuance Split

At epoch t, the protocol mints v_t tokens and divides them as:

$$v_t \;=\; \gamma_{\mathsf{CP}}\, v_t \;+\; \gamma_{\mathsf{V}}\, v_t \;+\; \gamma_{\mathsf{B}}\, v_t, \quad \gamma_{\mathsf{CP}} + \gamma_{\mathsf{V}} + \gamma_{\mathsf{B}} = 1.$$

- **Layer-2 with centralized sequencer**: $\gamma_{\mathsf{B}} = 0$, so all tokens flow to Compute Providers and Verifiers.
- **Layer-1 or decentralized proposers**: $\gamma_{\mathsf{B}} > 0$, block proposers receive $\gamma_{\mathsf{B}} v_t$, then share further with sub-providers using the same weighting.

Each actor's inflationary reward is:

$$R_{\mathsf{infl}}(i,t) = \gamma_X \, v_t \;\times\; \frac{\mathsf{Rep}(i,t)\,\mathsf{Contribution}(i,t)\,\mathsf{Stake}(i,t)}{\sum_{j\in\mathcal{W}_X} \mathsf{Rep}(j,t)\,\mathsf{Contribution}(j,t)\,\mathsf{Stake}(j,t)},$$

where $X \in \{\mathsf{CP}, \mathsf{V}, \mathsf{B}\}$ and $\mathcal{W}_X$ is the corresponding actor set.

Let $\Phi(t)$ be total fees collected in tokens, and q the fraction for rewards:

$$F_t = q\,\Phi(t).$$

We allocate F_t across Compute Providers and Verifiers by:

$$R_{\mathsf{fees}}(i,t) = F_t \;\times\; \frac{\mathsf{Rep}(i,t)\,\mathsf{Contribution}(i,t)\,\mathsf{Stake}(i,t)}{\sum_{j\in\mathcal{W}_{\mathsf{CP}\cup\mathsf{V}}} \mathsf{Rep}(j,t)\,\mathsf{Contribution}(j,t)\,\mathsf{Stake}(j,t)}.$$

Summing both components:

$$\mathsf{Rewards}(i,t) = R_{\mathsf{infl}}(i,t) + R_{\mathsf{fees}}(i,t).$$

7.2 Worked Example

Consider the following values. We use FLOP count in this example.

- Epoch t: $v_t = 10{,}000$, $\gamma_{\mathsf{CP}} = 0.7$, $\gamma_{\mathsf{V}} = 0.3$.
- Fee revenue $\Phi(t) = 2{,}000$, $q = 0.5$.
- Provider A: $\mathsf{Rep} = 0.9$, FlopCount $= 10^{12}$ of 3×10^{12} total, $\mathsf{Stake} = 0.5$.

Hence, the inflationary share is given by

$$R^{\mathsf{CP}}_{\mathsf{infl}}(A) = 0.7 \times 10{,}000 \times \frac{0.9 \times \frac{10^{12}}{3\times 10^{12}} \times 0.5}{0.15} \approx 7{,}000.$$

Fee-driven:

$$F_t = 1{,}000, \quad R_{\mathsf{fees}}(A) = 1{,}000 \times \frac{0.9 \times \frac{1}{3} \times 0.5}{\sum(\dots)}.$$

Total reward is the sum of these.

7.3 Remark on L1 vs. L2

On a fully decentralized L1, block-proposer rewards $\gamma_{\mathsf{B}} v_t$ incentivize honest proposal. In L2, with a single sequencer, $\gamma_{\mathsf{B}} = 0$, and the sequencer's revenue is treated off-protocol or re-distributed via dedicated governance.

7.4 Stake and Reputation

The final factors—$\mathsf{Rep}(i,t)$ (Section 5) and $\mathsf{Stake}(i,t)$—ensure that enduring trust and economic skin in the game modulate absolute work volume. A highly staked, high-reputation provider earns a larger share, so that anyone gaming FLOP counts or staking volatility is naturally disfavored.

The Rewards Mapping layer completes the economic feedback loop: inflationary issuance and fees determine the total pool ($\mathsf{base}(t)$), while reputation, verifiable compute contribution, and staked capital modulate each actor's share. This unified mechanism ties back to verifiable compute (Section 3), orchestrated job assignments (Section 4), and dynamic reputation (Section 5), ensuring that every token minted or fee collected reinforces honest, high-quality, and well-staked participation in Impulse's decentralized AI compute marketplace.

8 Token Economy

The native token underpins every layer of the *Impulse* protocol: it is the unit of exchange for jobs, the collateral staked in verifiable compute (Sections 3 & 6), the metric of governance weight, a utility (gas) token, and the denominated reward unit (Section 7). In this Section, we deepen the token-flow model, derive precise supply dynamics, and show how deflationary levers, staking schedules, and on-chain governance combine to sustain long-term value and align incentives.

8.1 Roles and Flow of the Native Token

At any epoch t, tokens flow through three primary channels:

1. **Inflationary Issuance**. The protocol mints M_t new tokens each epoch to reward block proposers, verifiers, and standby services ($\beta_{\text{block}}, \beta_{\text{sub}}, \beta_{\text{standby}}$ fractions; see Section 7.1).
2. **Transactional Fees & Treasury**. Users pay fees $\Phi(t)$ in native tokens when submitting compute jobs (Section 7.2). A governance-set fraction q of these fees flows directly into the reward pool. This reward pool will later be distributed across CP that provide the compute work necessary for the protocol; the remainder is stored in a *community vault* in the protocol, for which the community –and not the protocol– decide how to spend those funds (e.g., funding new grants, etc). These tokens are temporarily taken out of circulation, and denoted by B_t
3. **Staking and Lock-Up**. Compute Providers and Verifiers lock L_t tokens as collateral when they join *Impulse*, depending on their compute power. These tokens are unlocked once a CP decides to stop participating on the network.

8.2 Supply Evolution

Combining issuance, treasury, locking, and unlocking yields the discrete supply update:

$$S_{t+1} = S_t \; + \; M_t \; - \; \Delta B_t \; - \; L_t \; + \; U_t,$$

where M_t is the minting of rewards, ΔB_t are treasury tokens, L_t are the newly locked collateral, and U_t correspond to tokens unlocked from prior locks.

Define the **effective circulating supply** $C_t = S_t - L_t$. Then, net inflation per epoch is

$$\frac{C_{t+1} - C_t}{C_t} = \frac{M_t - \Delta B_t - (L_t - U_t)}{C_t}.$$

Governance adjusts p_{burn} and the issuance schedule $\{M_t\}$ (e.g. linear, decaying, or algorithmic) so that net inflation remains within a target band $\pi_{\min} \le \pi_t \le \pi_{\max}$.

8.3 Lock-Up Schedules

Collateral lock-ups (are deposited by CP as a way of guaranteeing honest behaviors) when a CP joins the network, they must stake an amount $s_{i,t}$ (proportional to their compute power) which is locked as long as the CP decides to participate in the protocol.

8.4 Governance and Parameter Adaptation

All key parameters—issuance schedule $\{M_t\}$, treasury allocation rate, fee split q, reward fractions $\beta_\star$, and lock-up durations Δ_{job}, Δ_{penal}—are subject to on-chain governance by native token holders. Proposals are passed via weighted voting, proportional to stake and reputation (Sections 5 & 7). To avoid rapid "fork-and-vote" attacks, parameter changes adopt a **two-stage timelock**:

1. **Signal Stage**: Preliminary vote signals community sentiment without effect.
2. **Execution Stage**: after a fixed delay, a final vote enacts changes on-chain, giving markets time to adjust.

8.5 Interplay with Compute and Reputation

The Token Economy completes the loop: verifiable compute (Section 3) and job scheduling (Section 4) generate on-chain fees and proofs that feed into the Rewards Mapping (Section 7), which mints M_t and distributes to staked, reputable providers. Reputation and Stake then determine scheduling priority and lock-up benefits, reinforcing honest, high-quality participation.

9 Minting

We adopt a goal-oriented mechanism as described in [40]. In particular, the core idea behind this mechanism is to

1. propose a set of observable, measurable (on-chain), quantities of interest — commonly known as Key Performance Indicators (KPIs), together with some time-dependent target values for these metrics, and

2. Adjust the minting rate according to how close (or far) these network parameters are from their target, in such a way that a *positive sum game is created*. That is, in a way, so that token recipients benefit whenever the network benefits.

We now formalize these ideas. Suppose we have N different KPIs. Let $\Theta \subset \mathbb{R}^N$ be the set of all possible states of our KPIs, let $\theta_t \in \Theta$ represent a specific state of these indicators at any given time t, and let $\theta_t^* \in \Theta$ be the vector of these target values at time t. We consider that to each KPI i there corresponds a (relative) level of importance $w_i \geq 0$ with $\sum_i^N w_i = 1$. This gives us a way of *favoring* one KPI over the other (or weight them all equally as $w_i = 1/N,\ \forall i = 1, .., N$).

Furthermore, for any $i = 1, 2, \ldots, N$, let $\delta_i : \mathbb{R}^2 \to [0, 1]$ denote an arbitrary measure of *distance* between $\theta_{i,t}$ and $\theta_{i,t}^*$ at time t. Notice that here I am using the term "distance" in a very loose way. Here each δ_i is non-decreasing with $z = \theta_{i,t} - \theta_{i,t}^*$ and has the property that $\delta_i(\theta_{i,t}, \theta_{i,t}^*) = 0,\ \forall\ \theta_{i,t} \geq \theta_{i,t}^*$. Furthermore, we define the N-dimensional distance $\delta : \Theta^2 \to [0, 1]$ as

$$\delta(\theta_t, \theta_t^*) := \sum_{i=1}^{N} w_i \delta_i(\theta_{i,t}, \theta_{i,t}^*) \tag{2}$$

Notice then that under our formulation each specific KPI has its distance function δ_i and (time-dependent) target value $\theta_{i,t}^*$.

Lastly, let $\rho_m, \rho_M \in \mathbb{R}_{\geq 0}$ denote the minimum (possibly 0) and maximum minting rates at any moment in time. These upper and lower bounds provide some *safety* rails so that (i) *at least some* tokens are minted when things are not going well or (ii) we don't over mint when things are going well. For any fixed t, we can then define the *instantaneous minting rate* as:

$$\rho(\theta_t, \theta_t^*) = \rho_m + [\rho_M - \rho_m] \cdot (1 - d(\theta_t, \theta_t^*)) \tag{3}$$

Notice then that, as the network reaches its KPIs (i.e., $\delta \approx 0$), we have $\rho \approx \rho_M$. Conversely, when the network is lagging behind its KPIs, $\rho \approx \rho_m$.

The formulation above thus induces a (simple) goal-adaptive minting mechanism: If tokens are being minted at every epoch τ (e.g., every $M \geq 1$ blocks), thus, the network mints $\rho(\theta_\tau, \theta_\tau^*)$ tokens until it runs out of tokens to distribute (if at all).

9.1 On the Choice of Metrics

The crux is then to find appropriate KPIs. Rather than track dozens of vanity statistics, we select at most **four** core KPIs that drive sustainable growth and resist manipulation:

1. **Verified Compute Volume** Measured as the total Number of FLOPs (or GPU-hours) successfully attested via our verifiable-compute protocols each epoch. This directly rewards network utility and is difficult to inflate—any false claims are caught and slashed.

2. **Active High-Reputation Providers** The count of distinct providers whose on-chain reputation exceeds a threshold (e.g. $\mathsf{Rep}_i > 0.8$) and who completed at least one job in the period. By focusing on quality-weighted participation, we incentivize both growth and reliability, making Sybil farming prohibitively costly.

3. **Fee Revenue Growth** The percentage increase in on-chain fee revenue $\Phi(t)$ compared to a moving average. As real users drive demand, higher fees signal healthy adoption; because fee payments burn or vest tokens, this metric inherently ties net inflation to actual usage.

4. **Job Success Rate** The fraction of submitted jobs that complete all verifiable-compute checks without slashing or timeouts. A high success rate reflects network robustness and a usable user experience; providers cannot game this metric without suffering economic penalties for failed tasks.

Each KPI i has a target curve $\theta^*_{i,t}$ that may follow a sigmoid adoption model—fast initial growth tapering to steady state—avoiding brittle exponential baselines. We define

$$\delta_i(\theta_{i,t}, \theta^*_{i,t}) = \max\left(0,\ \frac{\theta^*_{i,t} - \theta_{i,t}}{\theta^*_{i,t}}\right),$$

which vanishes when the actual value meets or exceeds the target and scales linearly if it is below. Weighting each w_i equally or by strategic importance ensures that no single metric dominates issuance.

10 Usability and Roadmap

Impulse AI is designed not just as a powerful on-chain compute protocol, but as a developer-friendly platform that anyone can adopt in minutes. Today, customers have two integration paths:

1. they can interact directly with the Impulse smart contracts—submitting jobs, staking tokens, and monitoring reputation—or,

2. more commonly, they can use our high-level SDK, RESTful APIs, and web interface.

In practice, most users will invoke the Impulse SDK, which streamlines the entire workflow: uploading datasets, selecting a pre-trained model, tuning hyperparameters, and dispatching a fine-tuning job with a few lines of code.

We believe that exceptional developer tooling is as critical as a robust protocol core. Accordingly, Impulse AI will underwrite and collaborate with third-party teams to build extensions, IDE integrations, and specialized dashboards atop our SDK. A dedicated grants program will fund companies creating value-add tools.

Looking forward, the SDK and UI will grow to support reinforcement-learning workflows, classical deep-learning pipelines, and full pre-training loops in addition to fine-tuning. We will layer in inference capabilities—enabling retrieval-augmented generation, multi-agent decision systems, and real-time chat interfaces—so that one unified SDK covers the entire

model lifecycle. Ultimately, we envision an AI assistant embedded in our console —a conversational agent that guides users through data preparation, model selection, deployment, and monitoring, all without requiring handcrafted code.

Beneath this developer surface lies a two-pronged technical roadmap. On the protocol side, we will expand our verifiable compute primitives to include zero-knowledge proofs that hide raw data while guaranteeing step-by-step correctness, support fully heterogeneous distributed training across arbitrary GPU topologies, and implement hardware-bound attestations that prove the exact GPU performed the work. We will integrate Trusted Execution Environments and homomorphic encryption to enable privacy-sensitive and regulatory-compliant training and inference. Mirroring these advances, the inference protocol will gain on-chain proofs of faithful execution, distributed low-latency inference orchestration, and privacy-preserving pipelines for user data.

Simultaneously, our developer toolchain will evolve in lockstep. We will introduce a custom model registry, allowing teams to import and version their private or proprietary architectures alongside open-source foundations. Multi-modal support will enable images, audio, video, and structured data tasks through the same simple API patterns. Deeper integration with Jupyter notebooks will allow inline proof verification, real-time metric visualization, and interactive debugging. Built-in model evaluation suites will automatically run benchmarks, performance profiles, and fairness audits during both training and inference. Finally, as AI assistants mature, our UI will offer chat-based setup wizards, cost-optimization advisors, and governance proposal drafting aids—making Impulse not only the most rigorous verifiable compute platform, but also the most delightful and productive environment for AI developers.

IMPORTANT NOTICE: DISCLAIMER

This Whitepaper is for informational purposes only and does not constitute a prospectus, an offer document, an offer of securities, a solicitation for investment, or any offer to sell any product, item, or asset (whether digital or otherwise). Furthermore, this document does not constitute legal, financial, tax, or other professional advice.

The information presented in this Whitepaper is subject to change or update without notice. *Impulse AI* makes no representations or warranties, express or implied, as to the accuracy or completeness of the information contained herein, and expressly disclaims any and all liability that may be based on such information or errors or omissions thereof.

This Whitepaper contains forward-looking statements. These forward-looking statements are not guarantees of future performance and are subject to risks, uncertainties, and other factors, some of which are beyond the team's control and could cause actual results to differ materially from those expressed or implied by these forward-looking statements. *Impulse AI* is currently in development and its final structure and functionality may differ from what is described in this document.

References

[1] Akash: Gpu pricing and availability. https://akash.network/pricing/gpus/ (2025), online Article. Accessed 03.12.2025

[2] Assran, M., Loizou, N., Ballas, N., Rabbat, M.: Stochastic gradient push for distributed deep learning. In: International Conference on Machine Learning (2019)

[3] Bellachia, A.A., Bouchiha, M.A., Ghamri-Doudane, Y., Rabah, M.: Verifbfl: Leveraging zk-SNARKs for a verifiable blockchained federated learning. arXiv preprint arXiv:2501.04319 (2025)

[4] Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M.S., et al.: On the opportunities and risks of Foundation Models. arXiv preprint arXiv:2108.07258 (2021)

[5] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., et al.: Language Models Are Few-Shot Learners. Advances in Neural Information Processing Systems (NeurIPS) (2020)

[6] Buterin, V.: Proof of Stake: The Making of Ethereum and the Philosophy of Blockchains. Seven Stories Press (2022)

[7] Chainlink: Chainlink: A decentralized oracle network. https://research.chain.link/whitepaper-v1.pdf (2017), accessed: 2025-02-19

[8] Chainlink Labs: Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. In: Whitepaper (2021), URL https://research.chain.link/whitepaper-v2.pdf

[9] Chen, B.J., Waiwitlikhit, S., Stoica, I., Kang, D.: ZKML: An optimizing system for ML inference in zero-knowledge proofs. In: Proceedings of the Nineteenth European Conference on Computer Systems, pp. 560–574 (2024)

[10] Choi, K., Manoj, A., Bonneau, J.: Sok: Distributed randomness beacons. In: 2023 IEEE Symposium on Security and Privacy (SP), pp. 75–92, IEEE (2023)

[11] Diskin, M., Bukhtiyarov, A., Ryabinin, M., Saulnier, L., Lhoest, Q., Sinitsin, A., Popov, D., Pyrkin, D., Borzunov, A., Wolf, T., Pekhimenko, G., et al.: Distributed deep learning in open collaborations. In: Advances in Neural Information Processing Systems (2021)

[12] Douillard, A., Donchev, Y., Rush, K., Kale, S., Charles, Z., Garrett, Z., Teston, G., Lacey, D., McIlroy, R., Shen, J., et al.: Streaming DiLoCo with overlapping communication: Towards a distributed free lunch. arXiv preprint arXiv:2501.18512 (2025)

[13] Douillard, A., Feng, Q., Rusu, A.A., Chhaparia, R., Donchev, Y., Kuncoro, A., Ranzato, M., Szlam, A., Shen, J.: DiLoCo: Distributed low-communication training of language models. arXiv preprint arXiv:2311.08105 (2023)

[14] Foundation, T.E.: Academic grants round 2025 wishlist (2025), URL https://efdn.notion.site/Academic-Grants-Round-2025-Wishlist-17bd9895554180f9a9c1e98d1eee7aec, accessed: 2025-03-06

[15] Gensyn: Gensyn Whitepaper: Decentralized infrastructure for distributed machine learning. https://gensyn.ai/whitepaper.pdf (2023), accessed: 2025-02-19

[16] Golem Factory GmbH: The Golem project: Crowdfunding a decentralized computing network (2016), URL https://cdn.prod.website-files.com/62446d07873fde065cbcb8d5/62446d07873fdeb626bcb927_Golemwhitepaper.pdf

[17] Google: Vm instance pricing. https://cloud.google.com/compute/vm-instance-pricing?hl=en#section-5 (2025), online Article. Accessed 03.12.2025

[18] Hagerup, T., Rüb, C.: A guided tour of chernoff bounds. Information Processing Letters **33**(6), 305–308 (1990), doi:10.1016/0020-0190(90)90013-F

[19] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al.: LoRA: Low-rank adaptation of large language models. ICLR **1**(2), 3 (2022)

[20] Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M., Chen, Z., Wu, Y., et al.: Gpipe: Efficient training of giant neural networks using pipeline parallelism. In: Advances in Neural Information Processing Systems (2019)

[21] Jaghouar, S., Ong, J.M., Hagemann, J.: OpenDiLoCo: An open-source framework for globally distributed low-communication training. arXiv preprint arXiv:2407.07852 (2024)

[22] Jia, H., Yaghini, M., Choquette-Choo, C.A., Dullerud, N., Thudi, A., Chandrasekaran, V., Papernot, N.: Proof-of-Learning: Definitions and practice. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1039–1056, IEEE (2021)

[23] Jung, S., Kim, H., Kim, Y.: zkCNN: Zero-knowledge proofs for convolutional neural network inference. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (2021)

[24] Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., et al.: Advances and open problems in federated learning. Foundations and Trends in Machine Learning (2021)

[25] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium, pp. 1353–1370 (2018)

[26] Keršič, V., Karakatič, S., Turkanović, M.: On-Chain Zero-Knowledge Machine Learning: An overview and comparison. Journal of King Saud University-Computer and Information Sciences p. 102207 (2024)

[27] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[28] Kulkarni, M., Gupta, N., Delworth, S.: Decentralized training of foundation models in heterogeneous environments. arXiv preprint arXiv:2206.01288 (2022)

[29] Lavin, R., Liu, X., Mohanty, H., Norman, L., Zaarour, G., Krishnamachari, B.: A survey on the applications of zero-knowledge proofs. arXiv preprint arXiv:2408.00243 (2024)

[30] LeCun, Y.: The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (1998)

[31] Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated Learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine (2020)

[32] Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., Harvilla, M.: A Proof of Useful Work for Artificial Intelligence on the Blockchain. arXiv preprint arXiv:2001.09244 (2020)

[33] Lin, J., Song, C., He, K., Wang, L., Hopcroft, J.E.: Nesterov accelerated gradient and scale invariance for adversarial attacks. arXiv preprint arXiv:1908.06281 (2019)

[34] Lin, Y., Han, S., Mao, H., Wang, Y., Dally, W.J.: Deep gradient compression: Reducing the communication bandwidth for distributed training. In: International Conference on Learning Representations (2018)

[35] Liu, M., Wong, H.S.P.: The path to a 1-trillion-transistor gpu: Ai's boom demands new chip technology. IEEE Spectrum **61**(7), 22–27 (2024)

[36] Liu, Q., et al.: Proof-of-Useful-Work: A consensus protocol for decentralized machine learning. https://arxiv.org/abs/1909.07962 (2019)

[37] Liu, S.T., Yu, J., Steeves, J.: Poster: Solving the free-rider problem in Bittensor. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, pp. 5045–5047 (2024)

[38] Madrigal-Cianci, J.P.: Analysis of filecoin gas: Modelling and uncertainty quantification (2023), URL https://www.youtube.com/watch?v=pRb1BSPZvRk, accessed: 2025-03-06

[39] Madrigal-Cianci, J.P.: Utility and demand for block space (2023), URL https://hackmd.io/LQ8Um2zURFeoGtjoGpiVyA?both=, accessed: 2025-03-06

[40] Madrigal-Cianci, J.P.: Bullish Minting (2025), URL https://hackmd.io/1_Ipeo87R8eE6JO631CniA?view, blog post. Consulted on March 4, 2025

[41] McMahan, H.B., Moore, E., Ramage, D., Hampson, S., Agueray Arcas, B.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of AISTATS (2017)

[42] Offchain Labs: Arbitrum Rollup: Scalable, private smart contracts. https://developer.offchainlabs.com/docs/intro (2021), accessed: 2025-02-19

[43] Parno, B., Howell, A., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252, IEEE (2013), doi:10.1109/SP.2013.31

[44] Perhats, M., Ferreira, M.X., Cortes-Cubero, A., Karra, K.: Local protocol (2024), URL https://palette-labs-inc.github.io/, accessed: 2025-03-06

[45] Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: Zero: Memory optimizations toward training trillion parameter models. https://arxiv.org/abs/1910.02054 (2020), arXiv:1910.02054

[46] Rao, Y., Steeves, J., Shaabana, A., Attevelt, D., McAteer, M.: Bittensor: A peer-to-peer intelligence market. arXiv preprint arXiv:2003.03917 (2020)

[47] Sergeev, A., Del Balso, M.: Horovod: Fast and easy distributed deep learning in tensorflow. https://arxiv.org/abs/1802.05799 (2018), arXiv:1802.05799

[48] Shoeybi, M., Patwary, M.P., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-lm: Training multi-billion parameter language models using model parallelism. https://arxiv.org/abs/1909.08053 (2019), arXiv:1909.08053

[49] Sun, H., Li, J., Zhang, H.: zkllm: Zero knowledge proofs for large language models (2024)

[50] Team, T.A.: Aethir whitepaper. https://3028335560-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2FlJdZs7NyMJ6Ewm4U1eRP%2Fuploads%2FOVdpd7QoNIDAZdfpGrGV%2FAethir%20Whitepaper.pdf?alt=media&token=ec38bfde-1668-472d-97d7-a48fb1300703 (2024), online Article. Accessed 03.17.2025

[51] Team, T.G.: Gensyn litepaper. https://docs.gensyn.ai/litepaper (2022), online Article. Accessed 03.17.2025

[52] Team, T.H.: Heurist whitepaper. https://www.heurist.ai/whitepaper (2024), online Article. Accessed 03.17.2025

[53] Team, T.P.I.: Introducing prime intellect's protocol & testnet: A peer-to-peer compute and intelligence network. https://www.primeintellect.ai/blog/protocol (2024), online Article. Accessed 03.17.2025

[54] Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains. In: Aspects of Computation and Automata Theory with Applications, pp. 377–424, World Scientific (2024)

[55] Thaler, J., et al.: Proofs, Arguments, and Zero-Knowledge. Foundations and Trends® in Privacy and Security **4**(2–4), 117–660 (2022)

[56] Wang, S., et al.: BlockFL: A blockchain-based federated learning framework. In: IEEE International Conference on Communications (ICC) (2020)

[57] Wikipedia: 2020–2023 global chip shortage (2023), URL https://en.wikipedia.org/wiki/2020%E2%80%932023_global_chip_shortage, accessed: 2025-03-06

[58] Woodhead, P., Cortes-Cubero, A.: Checker network protocol: Litepaper (2025), URL https://blog.checker.network/posts/checker-network-litepaper-1, accessed: 2025-03-06

[59] Woodhead, P., Team, T.C.: Checker network protocol: Economic white paper (2025), URL https://bafkreiaw4vizzgufxse2bohdzsgt57zvvlduajekm4n6ljh6l5hf2tg5iq.ipfs.w3s.link/, accessed 2025-05-06

[60] Xu, L., Zhang, B., Chen, W.: VERITAS: A framework for verifiable federated learning. Proceedings of the 40th International Symposium on Reliable Distributed Systems (2021)

[61] Zhang, Y., Wang, S.: Proof of sampling: A nash equilibrium-secured verification protocol for decentralized systems. arXiv preprint arXiv:2405.00295 (2024)

[62] Zhao, Z., Fang, Z., Wang, X., Chen, X., Su, H., Xiao, H., Zhou, Y.: Proof-of-Learning with incentive security. arXiv preprint arXiv:2404.09005 (2024)

[63] Zuo, X., Wang, M., Zhu, T., Zhang, L., Ye, D., Yu, S., Zhou, W.: Federated trustchain: Blockchain-enhanced llm training and unlearning. https://arxiv.org/abs/2406.04076 (2024), arXiv:2406.04076

Appendix

A The State of Verifiable, Decentralized Training

In what follows, we present a brief literature review of the state of the art on verifiable compute (Section A.1) and distributed training (Section A.2) in the context of decentralized training of large-scale machine learning models. For brevity, the presented review is intentionally short. However, we refer the interested reader to [22, 61, 62, 3, 29] and [45, 34, 13, 12] (and the references therein) for comprehensive surveys of recent methods in verifiable and distributed computing, respectively. We also note that several emerging whitepapers, including those by Heurist.ai [52], Aethir [50], Gensyn [51], Akash [1], and PrimeIntellect [53], offer conceptual frameworks for trustless or decentralized AI compute. While these proposals vary in their specific architectures and token-based economic models, most acknowledge the core challenge of verifying correctness in large-scale training tasks and the need for techniques that mitigate overhead without sacrificing security.

A.1 Verifiable Compute

We begin by defining some essential concepts. Let T denote a training task parameterized by model weights W and hyper-data D_T (encompassing hyper-parameters and training data). A *verifiable compute* (VC) mechanism in our setting comprises three components. First, there is a mapping A that satisfies $W_{\text{new}} = A(W_{\text{old}}, D_T)$, representing a training step. Second, there is a *proof certificate* Π_T (e.g., a zero-knowledge proof or transcript) that shows how W_{new} was obtained from W_{old} using D_T. Finally, there is a *verification procedure* $V(\Pi_T, W_{\text{old}}, W_{\text{new}}, D_T)$, which returns true if the proof Π_T is valid. The goal is to ensure that the trainer has indeed performed the requisite work to produce the update W_{new}. Typical properties of VC mechanisms are *completeness* (an honest trainer can produce a proof that verifiers accept), *soundness* (no adversary can pass off a wrong model update as correct), and *relative efficiency* (the computational cost of generating and verifying proofs remains small compared to training itself) [22].

Although these properties establish a theoretical foundation for verifiable computation, current approaches do not efficiently scale to decentralized large language model (LLM) training and fine-tuning. Both cryptographic proofs (e.g., zero-knowledge systems) and alternative trust methods (spot-checking, redundant execution, or proof-of-learning transcripts) can incur substantial overhead or become insecure at large scales, as discussed below.

A.1.1 Cryptographic Proofs (zkML)

One prominent approach is to rely on cryptographic proofs, often zero-knowledge succinct arguments (zk-SNARKs and STARKs [29]), to certify that training computations were carried out correctly. In principle, such proofs can show that a large-scale computation (such as a transformer training step) has been executed faithfully, with the verification time remaining independent of the model's size [55]. While this provides strong correctness guarantees, compiling even a modest neural network training phase into a SNARK circuit is expensive. Recent analyses indicate multi-order-of-magnitude slowdowns just for inference in a zero-knowledge setting [9], making full-scale LLM training prohibitively expensive. For instance, VerifBFL [3] demonstrated verifiable federated learning by generating zk-SNARK proofs for each participant's local training. The on-chain verification step was efficient (under one second), but creating a proof even for simple convolutional networks took on the order of a minute per batch. Scaling these techniques to 70B-parameter models without a major breakthrough in proof efficiency is currently not feasible.

Projects such as Heurist.ai, Gensyn, and others occasionally mention zero-knowledge or multi-party approaches as long-term goals for verifiable compute, but they too acknowledge the practical barriers of zk-circuit overhead when handling large models. Gensyn, for example, proposes partial replication of training steps coupled with watchers that replicate small portions of the execution, citing the intractability of fully cryptographic approaches for massive LLM workloads.

A.1.2 Proof-of-Learning (PoL)

The Proof-of-Learning (PoL) concept [22] exploits the idea that the sequence of weight updates generated by standard optimizers (SGD or ADAM [27]) is hard to forge without doing the actual work. Under PoL, the trainer logs a series of intermediate states and hyperparameters. A verifier randomly replays selected segments to check if those model updates align with the claimed transitions. This random sampling can offer probabilistic security. Constructing a fake log that plausibly reproduces the final weights without honest training is generally as difficult as performing the training itself.

However, repeated spot-checks still incur nontrivial overhead. Recent extensions such as [62] and [61] suggest combining PoL with "capture-the-flag" incentives or random sampling protocols that penalize malfeasance with staked tokens. These proposals attempt to reduce average verification costs while discouraging trainer collusion. Yet they still remain challenging at the scale of LLM training, where each step involves billions of floating-point operations.

A.2 Decentralized Distributed Training

When training large language models, the computational demand is often immense due to the sheer parameter count and data volume. Distributing the workload across multiple geographically separated machines is a practical way to mitigate these expenses. Some whitepapers in the decentralized space, such as Aethir, reference enclaves to protect off-chain computation but do not fully detail how to orchestrate globally distributed training across many untrusted nodes. Gensyn's Litepaper proposes bridging off-chain GPU resources while leveraging partial replication and a custom scheduling layer, and Akash focuses on container-based resource leasing with less emphasis on verifiable large-scale updates. PrimeIntellect mentions distributed pipelines but still relies on stake-based assumptions for correctness.

From a more algorithmic standpoint, the distributed training literature is extensive. Douillard et al. [13, 12] introduced DiLoCo, a low-communication protocol that clusters updates from local workers and periodically averages parameters. By reducing the frequency of synchronization to every few hundred steps, DiLoCo can accommodate intermittent connectivity while preserving accuracy. Streaming DiLoCo [12] extends these ideas further by partially synchronizing and quantizing model updates. Such methods demonstrate that large-scale models can be trained without a dedicated high-speed interconnect, reducing bandwidth by factors of 100–500. Rajbhandari et al. [45] also show how careful partitioning and gradient accumulation strategies can yield efficient parallelism on commodity hardware.

When applying these distributed techniques in a decentralized setting, the remaining question is how to verify each participant's contribution to ensure correctness and detect cheating. Many whitepapers (Heurist.ai, Aethir, etc.) refer to building trust through enclaves, while others (Gensyn) propose partial re-computation or staking-based fraud proofs. Yet the overhead and security trade-offs become increasingly nontrivial at scale. A robust solution must combine low-communication distributed optimizers with a verification scheme that remains efficient over potentially billions of parameters.

Overall, although the distributed training literature offers multiple ways to reduce communication and accommodate heterogeneous hardware, integrating these approaches into a secure, trustless framework for large-scale verifiable training is still an open challenge. Our goal is to advance this frontier by leveraging, refining, and extending the PoL techniques, cryptographic proofs, or incentive-based verifiers, so that fully decentralized fine-tuning of large models becomes tractable in practice.