Decentralized Compute's Final Frontier

Solving the Verification Problem with Optimistic Systems and ZKPs

Distributed compute networks face a unique version of the verification problem. With zero-knowledge proofs still prohibitively expensive (yet important), optimistic solutions, enabled by HellasAI, promise quick deployment and low cost while increasing demand for decentralized compute networks.

Imagine this– it's 2030 and decentralized compute networks have won.

The Akash, Prodia, and IO.Nets of the world service billions of queries each day. Everything from Macy's AI shopping assistant (it's inevitable) to hacker houses in Lisbon indirectly use millions of whirring GPUs in decentralized data centers located around the world. Gensyn trains the latest models using the spare compute on my phone and your laptop all while earning us rewards seamlessly in the background. Even the big datacenter players–Microsoft, Google, and Amazon–offer their services on a competitive, open marketplace enabled by blockchains.

One little thing, in this world, no one solves the **verification problem**. At each step, everyone just trusts that the output they get is correct. After all, it's a big part of our current modus operandi; in this world, it continues to be.

This rosy reality will inevitably come crashing down. A lack of verification opens up an entire supply chain of potential attacks. Malicious actors could covertly affect model performance itself by poisoning data that finetunes the model or inserting sponsored ads into responses without disclosing it. In the worst case, the world's critical systems are flooded with a red tide of misinformation. And while an attack of this magnitude will invariably be noticed, the results in the meantime could be truly catastrophic. Think doctors, assisted by AI models, diagnosing and operating on imaginary tumors or military drone systems spontaneously opening fire on the wrong targets.

However, it's more likely that this world is impossible to even reach. Entrepreneurial compute providers, realizing they can offer less performant (and therefore less computationally intense) models all while claiming the same fee would be incentivized to fatten their wallets at the expense of the network. In fact, without a trustworthy verification method, no real distributed compute future can exist.

The verification problem for compute networks.

Almost all decentralized networks rely on a staking and slashing mechanism. Nodes will put up collateral or "stake," which consequently gets destroyed or "slashed" if the node acts improperly. This introduces an economic incentive for nodes to follow the protocol. A naïve way of implementing the slashing is for the protocol to designate some trusted entities that check the computation; this, however, entrusts a select few nodes with the protocol's safety, resulting in centralization and redundant computation. Protocols, therefore, utilize smart contracts known as "slashing contracts."

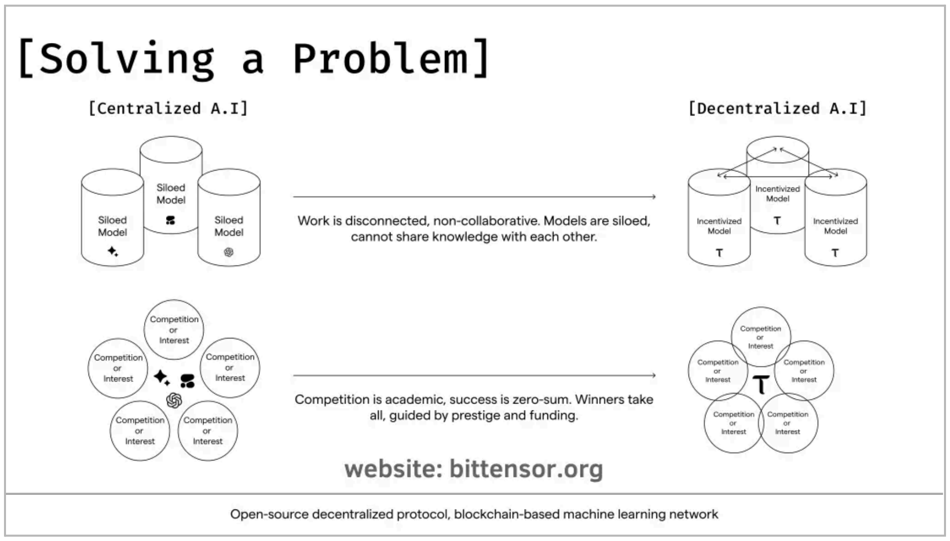
But what is actually being submitted to the slashing contract? Because each node must be able to append its local ledger to slash a deviating node's stake, each node must either be able to verify the fault themselves or be convinced that the other nodes acted in accordance with the protocol. The latter method can involve the use of zero-knowledge proofs (ZKPs).

Zero-Knowledge Proofs

ZKPs are cryptographic mechanisms that enable one party (the prover) to convince another party (the verifier) of the truth of a statement without divulging any details about the statement itself, except its validity. While the computational overhead of proofs is currently a limiting factor—cutting-edge solutions require four to six orders of magnitude more compute than running the program to generate the proof—the proofs are quick to verify for nodes. For context, six hours of magnitude means that in the time it takes to generate a proof on an Apple M3 Max chip, a Ti-84 graphing calculator could run the underlying program 150 times! Clearly there is still work to be done, but importantly, these proofs only need to be generated by one party, and techniques like nesting proofs within each other, often called recursive ZKPs, can enable constant time verification for all nodes on the network. When private models are run or a dataset is proprietary, for instance, ZKPs ensure that nodes verifying the computation cannot gain access to sensitive information.

ZKPs could also be a large part of unlocking idle compute on consumer hardware—a MacBook, for example, could use its extra compute bandwidth to help train a large-language model while earning rewards for the user. Deploying decentralized training or inference with consumer hardware is the focus of teams like Gensyn; for systems where the computation is sharded among nodes themselves (i.e., each node runs part of a computation and the result is then woven back together), ZKPs could provide an important verification that data was not leaked or that a malicious node didn't introduce error to mess up the final reconstructed computation. While ZKPs will undoubtedly get much faster to generate and serve a broader role than just helping distributed compute networks (Ethereum's single-slot finality, for example), they require further specialized research and development.

Given the current limitations of ZKPs, modern solutions like Bittensor, Inference Labs, and Hellas AI have devised unique solutions and architectures to solve the verification problem. Bittensor's architecture, however, precludes it from being a feasible solution for distributed compute networks.



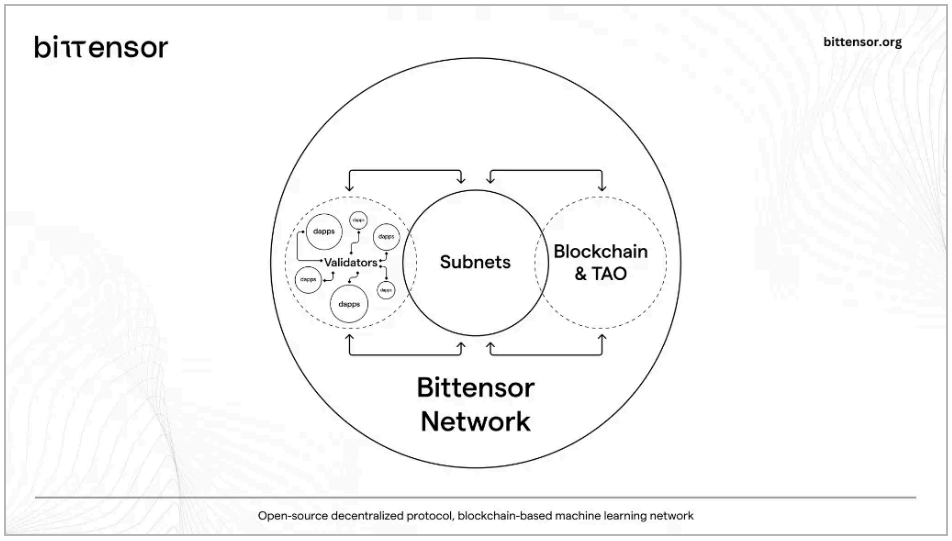
Work is disconnected, non-collaborative. Models are siloed, cannot share knowledge with each other.

Competition is academic, success is zero-sum. Winners take all, guided by prestige and funding.

Open-source decentralized protocol, blockchain-based machine learning network

The Goal: According to Bittensor, decentralized AI systems allow for much more dynamic competition and interaction between traditionally siloed systems.

Bittensor is composed of sub-networks containing nodes each running their own AI models. Each subnetwork is built for a specific use-case, and models are built to best serve those needs. When a user queries the subnet, each node runs the query on their model, and the "best" response is returned, determined by an evaluation function run by each subnet's validators. While the efficacy of any evaluation function to fully capture a user's preference is uncertain at best, it has also resulted in the gamification of models that seek to fulfill those criteria. For example, although Bittensor has improved the evaluation algorithm in their S1 subnet multiple times, the disproportionate success of one or two models has also created centralization pressure even with partial compensation for non-winning nodes. In theory, thousands of subnets for specific functions could enable more niche model developers to create specific use-cases and compete against larger providers. This sounds feasible and unique, but in practice could also advantage operators with large compute resources and sophisticated models, creating a centralization flywheel of sorts that it seeks to avoid. As larger providers, already advantaged by their size, begin to outperform their counterparts and receive greater compensation, they'll put that gain toward increasing their advantage.


bittensor
bittensor.org
dapps
Validators
Subnets
Blockchain & TAO
Bittensor Network
Open-source decentralized protocol, blockchain-based machine learning network

In Bittensor's model, each subnetwork offers its own service. Validators, who serve responses directly to applications, query and validate subnetwork outputs. Broadly, these subnetworks interface with the blockchain layer which handles the distribution of rewards and payment for services.

Bittensor's framework, however, does not work well for clients who need to run specific models, since a user query is evaluated separately by different models. For a compute network where clients need to run specific programs, this weighted-average method cannot be applied. This is because the output will not match the user's model since it includes parts from other models as well. However, if a Bittensor subnet sets its evaluation function to match a specific model, then the validator nodes would have to run the model themselves to determine the proper output, resulting in significant redundant compute. This feature distinguishes Bittensor from compute networks like Akash, Render, and Ionet. Whether an evaluation function can successfully determine output for a valuable use-case remains a topic of contentious debate.

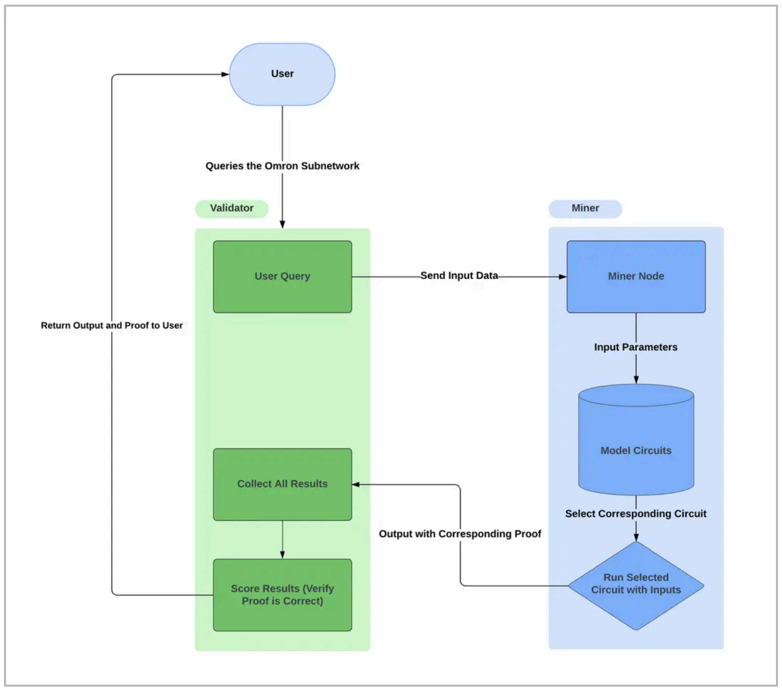
Because of their evaluation function, Bittensor natively solves the verification problem: nodes that skimp on compute simply won't perform as well. Each subnetwork is able to pick its reward distribution mechanism to fit its criteria. For some, like Subnet 9 which aims to produce state of the art pretrained LLMs, a winner-takes-all approach is used. Just allowing the "best" model to win, however, incentivizes losing nodes to plagiarize and fine-tune the existing winner rather than make meaningful improvements to their own. To ensure that developers meaningfully improve their models, subnets each set determine how much “better” a new model must be to be considered winning. But how much is correct. 1%? 5%? 10%? Aim too low and node operators can’t invest enough to truly improve their model since they’ll quickly be kicked off. Too high and it’s a centralization flywheel where one node stays in power, earns rewards, and widens their lead. Some subnetworks distribute compensation on a curve, rather than winner-takes-all, incentivizing nodes to keep developing.

Inference Labs—A Hybrid Approach

Inference Labs aims to offer a hybrid solution, but its use of ZKPs make it a longer term investment. Unlike Bittensor, Inference Labs has to ensure that each user request is computed correctly.

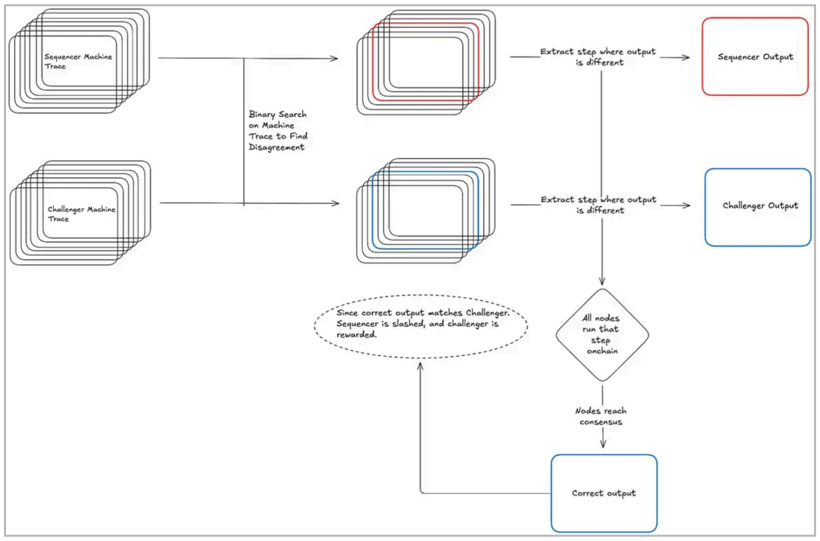
One part of their solution offsets the costs from a node having to generate a ZKP for each inference job completed. How? A user can deploy a model on the network by running a node, thereby committing compute capacity for that model. A user can query the model, and at any point, any member of the network can pay to issue a challenge if there is suspicion the result is incorrect. The challenge process requires the creation of a ZKP of the inference, then verified by the other nodes on the network. For mission-critical tasks, however, a user can challenge up-front therefore requiring a proof to be generated and guaranteeing that the node properly conducts the inference or is slashed. For private models—think, trying to query ChatGPT—only the node running inference can properly generate the proof.

Omron, a Bittensor compute cluster for incentivized proof generation, enables Inference Labs to outsource proof generation in the event of a challenge. Since the model is open-source and the network has access to the inference request, Omron enables any neuron in their Bittensor subnet to generate a ZKP of the computation. This works because the evaluation function the Bittensor validators check is just that the proof was produced correctly, incentivizing the smallest proofs in the fastest time. Using Omron allows Inference Labs to gain immediate access to compute for proofs while incentivizing miners and validators to join the network; in the future, this could be abstracted away to prover networks.


User
Queries the Omron Subnetwork
Validator
Miner
User Query
Send Input Data
Miner Node
Return Output and Proof to User
Input Parameters
Model Circuits
Collect All Results
Select Corresponding Circuit
Output with Corresponding Proof
Score Results (Verify Proof is Correct)
Run Selected Circuit with Inputs

Hellas: Optimistic Games

A different method is to employ optimistic games which are immediately deployable and don't require advances in zero-knowledge proofs. Perhaps most well-known from its deployment in the Optimism bridge challenge game, optimistic games incorporate game theory mechanics to promote accordance with the protocol. In an optimistic game, one party stakes some collateral and makes a claim, which could be the output of an inference request or the state of a rollup; during a challenge period, a node on the network can challenge the claim. What ensues is the "dispute game," a method of determining which party is correct. If the challenger is correct, they get a reward, creating an incentive to monitor the chain. At scale and with a high enough stake, this should result in full adherence to the protocol, because the economic risk of getting caught doesn't outweigh the marginal gain in value from skimping out. It's worth noting that this requires not only high enough negative externality from being slashed, but also a high enough probability that the computation will be run by another node and, most crucially, a method of conducting an optimistic game itself. Furthermore, because this requires those monitoring the network to be able to run the transactions, it currently is only implemented with open-source models on non-sensitive queries. For privacy-preserving applications, ZKPs still serve as the most feasible solution.


Sequencer Machine Trace
Binary Search on Machine Trace to Find Disagreement
Extract step where output is different
Sequencer Output
Challenger Machine Trace
Extract step where output is different
Challenger Output
Since correct output matches Challenger, Sequencer is slashed, and challenger is rewarded.
All nodes run that step onchain
Nodes reach consensus
Correct output

How an optimistic game works. For more details, see Appendix I

To enable an optimistic dispute game, Hellas uses a unique method of representing models—as a hypergraph data structure—that allows for easy extraction of the circuits to quickly generate ZKPs. This is combined with their representation of a model inference as a series of tensor operations to enable an optimistic game. A challenger disagreeing with a compute node can generate a trace (a series of tensor operations and associated states), find the specific cluster that led to disagreement, and generate a ZKP of the correct computation for that specific circuit. The proof is then validated on-chain, and an improperly acting node's stake is redistributed. While Hellas's own nodes will likely monitor the chain and conduct verification when the network initially goes live, the system is designed to move toward decentralization.

This method of representing the model and computations also has other positive externalities. Because nodes are running a series of tensor operations in Hellas's format, it enables Hellas to act as an aggregating layer and plug into extant decentralized compute networks seamlessly. For most developers, the cost of plugging into a compute network can be high, involving spinning up "containers" for their code according to the specifications of the individual networks themselves—this increases the cost of switching and adds significant barriers to entry. While Hellas nodes will be required to spin up an instance and deploy individually in each network, the end result is a seamless execution on the developer's end where the underlying compute infrastructure is abstracted away. In this way, Hellas is akin to Next.Js—a framework for writing react (a library widely used in web development) that simplifies many of the best practices in an easy-to-deploy solution that reduces the technological expertise required to make responsive websites. For a more comprehensive example, see Appendix III. While Next.Js is entirely open-source, the company that developed the framework, Vercel, was able to profit heavily by charging for the cloud services on which the websites run; as of May 2024, Vercel completed a $250M Series E round at a $3.25B valuation. Clearly, open-source can be profitable. Hellas, with its unique system and plans for node network, seems well-equipped to capture a similar market share for decentralized AI compute by shipping an easy-to-use plug-and-play option that abstracts away the difficulty for running AI models. The vision: deploy your model on a DePIN compute network with a few easy lines of code, as simple as plugging in your OpenAI API key.

In Sum

One of the fundamental challenges with distributed compute networks is the verification problem, on how the network can ensure nodes act in accordance with the protocol. While the Bittensor model does not suffer from this challenge, it's rather unclear if any evaluation function can effectively capture user preference or if the proliferation of many small-specialized models will enable smaller developers to grow and develop new tech. It's quite possible that a few large models will solve user preference and the marginal utility of a network won't justify its usage. However, as Inference Labs has shown by using Omron, Bittensor subnets can be used to incentivize quick ZKP generation. While their hybrid solution allows for verification, it relies quite heavily on ZKP systems, but they are working to engineer novel, faster systems tailored to machine learning inference. Given the amount of capital deployed into developing ZKP systems, it does seem likely that improvements will only continue to accelerate; to enable similar response time as centralized systems, however, will require much more development.

Hellas shows significant promise in their solution to the verification problem by enabling an optimistic challenge system. While the contested step is verified using a ZKP (or perhaps a succinct proof since the models are open-sourced), the ability to easily extract circuits that represent relatively few steps of computation rather than the entire inference significantly reduces the computation required for the proof. The ability to engage in a dispute game marks a shift in the prevailing wisdom surrounding distributed compute. And while, outside of crypto, open-source software struggles to gain moat, the unicorn valuation of Vercel shows that engineering seamless, easy to deploy systems that abstract away the difficulties of deployment has significant sticking power. Furthermore, Hellas's ease of use will likely drive greater demand for these compute networks– a problem that Akash, Ionet, and Render all contend with–while capturing value by potentially commoditizing the underlying layers.

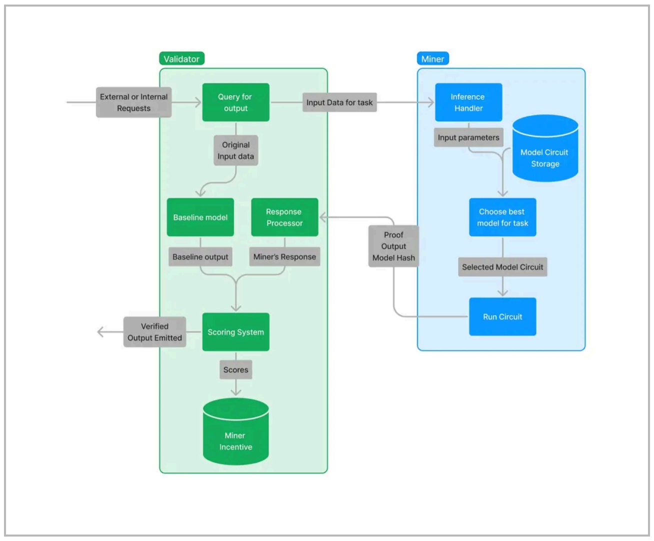
Being the plug-and-play solution, however, extends well beyond acting as an aggregation layer for compute networks. Given the sheer spend on land, chips, and energy infrastructure to build data centers in the past eighteen months, it's clear that compute has become an invaluable commodity. As AI systems are used by humans and soon other AI agents at a rapidly increasing rate, billions of inferences will be made each day. If Hellas can become the de-facto format for model representation, they not only will enable smaller datacenters and private servers to be trustlessly utilized by enterprise clients but also large network effects. Especially if deployment is as simple as using a traditional API key.

Hellas's optimistic system captures market share, drives most relying upon Hellas's format to deploy across a stack of compute networks. But even if growth isn't as aggressive, we firmly believe AI use will continue to proliferate and autonomous agents will act among us by the end of the decade. Given the sheer spend on land, chips, and energy infrastructure to build data centers in the past eighteen months, it is evident that compute has become an invaluable commodity. Hellas's optimistic dispute game for AI inference promises to unlock the idle processing power in smaller datacenters and private servers, enabling the vision of decentralized compute while making the developer experience as easy as plugging in an API key.

**** Appendix ****

I. For those interested, the dispute game for rollups works as follows: a sequencer pushes the state of L2 to Ethereum. Since the transaction data (smart contract calls and value transfers between accounts) has been made available using a DA platform, any node in the network can run the transactions, determine the new state, and compute the state root. If the state roots differ, they start the challenge game. While the mechanics are more complex due to the novelty of Optimism's challenge game where any node can "play" for either side, we'll assume that the original sequencer and challenger node play the game to its conclusion. Both sides engineer a machine trace, a sequence of transactions and corresponding state, and conduct a binary search until they find the discrepancy. Since the final states disagree, it follows that there exists at least one intermediary step where the transactions disagree; the binary search finds this discrepancy. That one step is run on-chain, and if the challenger wins, then the sequencer is slashed and the challenger is rewarded. This creates an economic incentive for nodes to double-check the state posted to the L2.

II. The Omron network can be extended to query models, here's what that looks like:



III. A tangible, technical example of the benefit of Next.Js is when using a react app a developer must introduce routing and have to install a react router. That entire component must then be wrapped with a router provider that then routes to the whole app. Rather than have a developer compare the options between router providers and react locations, Next.Js already comes with a router integration that's ready to use without any additional setup.