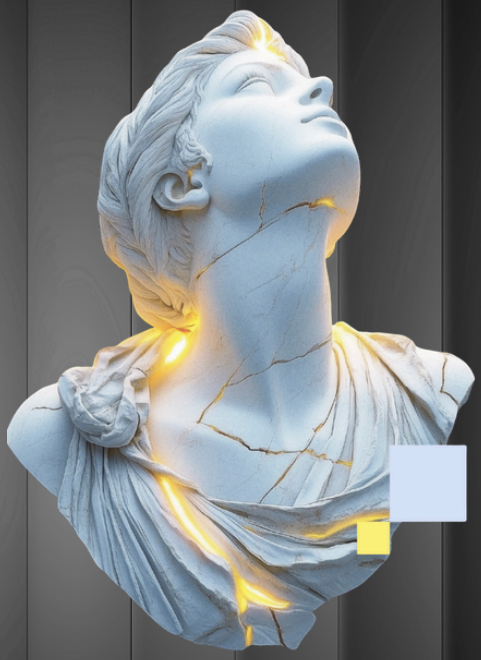
Lemur Labs
MESSAGER OF CLEAR IDEAS

HELLAS

State of Decentralised Compute



December 2025

Table of Contents

1. Inference Economics and the Emerging Limits of Hyperscale Cloud

According to Deloitte's 2026 Tech Trends report, the AI compute stack is quietly but decisively rebalancing. What once revolved around episodic training runs is giving way to inference, the continuous, real-time execution of models embedded directly into products, workflows, and user experiences.

By 2026, inference is expected to account for roughly two-thirds of all AI compute, up from just one-third in 2023, as generative and agentic systems move from experimentation into everyday operation. Crucially, this shift is unfolding even as per-token costs have collapsed, by an estimated 280-fold over the past two years, revealing a counterintuitive reality.

AI spending is no longer constrained by model efficiency, but by how often and how persistently models are used. Cloud bills, in turn, are increasingly shaped not by the one-time cost of training, but by the relentless cadence of inference itself. Viewed through this lens, the infrastructure assumptions that underpin traditional cloud compute begin to strain.

The very platforms that excelled at elastic, bursty workloads now face structural friction as inference becomes always-on, latency-sensitive, and economically continuous, setting the stage for a deeper re-architecture of where and how AI runs. This shift turns inference from a technical concern into a structural cost, reliability, and compliance problem for enterprises.

- **Cost inefficiency** at scale, cloud platforms are optimized for elastic, variable workloads, but inference costs accrue every time an AI system responds to a query, creating ongoing operational spend rather than one-time training expenditure.

- **Resource provisioning tension**, hyperscale data centers are increasingly saturated with AI workloads, and capacity constraints may intensify as AI's share of data-center demand grows.

- **Geo-latency and regulatory friction**, enterprise adopters are demanding local, low-latency execution and sovereign infrastructure options as inference becomes core to customer experience and compliance.

Cloud infrastructure was built for a world where computing happens in bursts. Capacity is rented, used intensively for a period of time, and then released. That model works well for training AI systems, which run as large, time-bound jobs. Inference behaves very differently.

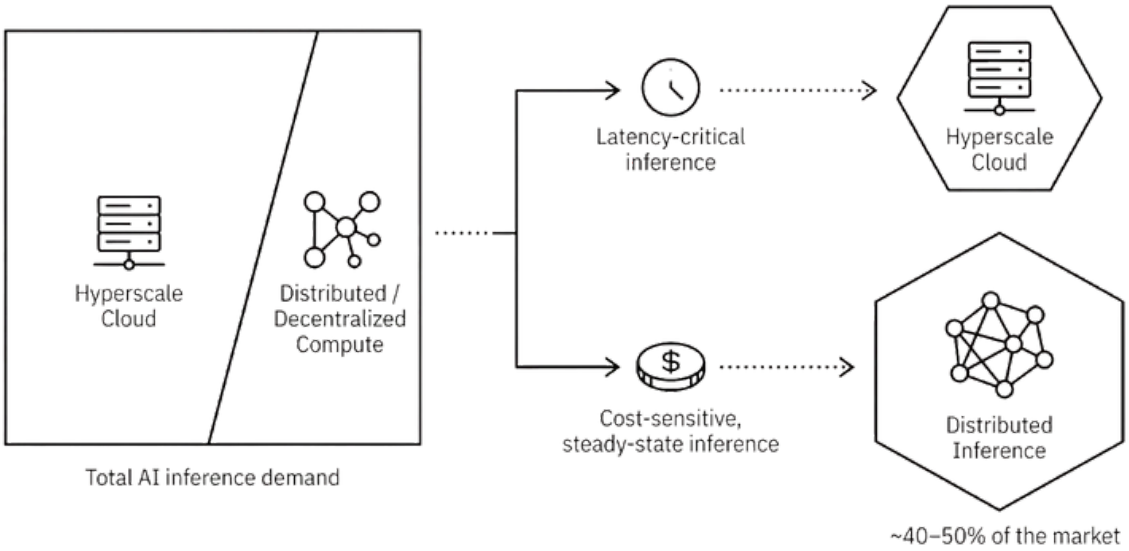
Once AI is embedded into products, chatbots, copilots, recommendations, automated decisions, it must remain available at all times. Models need to be kept in memory, ready to respond instantly, even when demand is uneven or temporarily low. Costs therefore accrue not when AI systems are actively performing computation, but simply by remaining provisioned and on standby.

This is where traditional cloud compute becomes increasingly ill-equipped to support the next phase of growth. Pricing and scheduling are still optimized around rented capacity rather than continuous usage. Providers must provision for peak demand, yet much of that capacity sits idle between requests.

As inference volume scales, this mismatch turns into a structural cost problem, spending rises even as models become cheaper and more efficient. What appears, on the surface, as a problem of infrastructure is in reality a problem of economics.

As this gap widens, a second execution layer begins to make sense. Decentralized compute allows inference to run closer to where demand originates, to tap underutilized hardware, and to separate always-on workloads from burst capacity. The result is not the displacement of hyperscale cloud, but a gradual bifurcation of the market.

Cloud remains the system of record and the default for elasticity and compliance. Steady-state inference gravitates toward execution models designed for cost efficiency, locality, and continuous availability. This shift is driven by usage economics rather than architectural experimentation, marking the next frontier of AI infrastructure investment. As a result, the infrastructure challenge shifts from accessing compute to coordinating consistent execution across heterogeneous and untrusted environments.



2. Why Inference Economics Favor New Execution Models

As AI systems move from isolated experiments to everyday infrastructure, the nature of compute demand changes in subtle but important ways. Inference is not just more compute. It is a different kind of workload, with different economic and operational pressures. These pressures explain why decentralized compute is not an alternative by ideology, but a natural fit by design.

3. Why Determinism Becomes Critical as AI Systems Begin to Act

The tolerance for nondeterminism in machine learning was shaped by an earlier generation of applications, recommendation systems, search ranking, and conversational interfaces, where variation in output carried little consequence. That tolerance erodes as AI systems evolve from passive responders into active agents.

Autonomous agents schedule tasks, execute trades, route customer interactions, trigger payments, and coordinate with other software systems. In these contexts, reproducibility becomes essential. If an agent behaves differently across identical inputs, failures can no longer be dismissed as statistical noise, they propagate downstream, trigger inconsistent actions, and undermine trust in the system as a whole.

In inference environments, this problem is magnified, slight numerical differences across hardware or runtimes prevent reliable caching, break unaltered execution, and make it impossible to reason about cost, latency, or failure recovery.

Deterministic execution does not constrain model intelligence, it constrains system behavior, allowing agent-based workflows to be audited, replayed, and safely composed.

As AI agents become embedded in financial operations, logistics, infrastructure management, and enterprise automation, determinism shifts from an academic concern to a prerequisite for deploying AI as reliable infrastructure rather than probabilistic tooling.

Inference naturally pushes computation closer to users and data

As AI becomes embedded in customer-facing products, internal tools, and autonomous systems, where inference runs begin to matter as much as how fast it runs. Deloitte highlights the growing importance of latency, data locality, and regulatory compliance as AI systems move into production, particularly in regulated industries and international markets.

Centralized cloud regions are optimized for efficiency at scale, but they are often geographically distant from users and data sources.

Decentralised computing offers a practical response. By enabling inference to run on-premise, at the edge, or across federated regional providers, it reduces latency and simplifies compliance without requiring enterprises to replicate full hyperscale infrastructure.

This pattern is familiar. Content delivery networks emerged for similar reasons when the web shifted from static pages to real-time, interactive experiences. Inference follows the same path outward, driven by proximity rather than raw scale.

Hyperscalers dominate early, but economics drive diversification over time

Deloitte’s report is states that hyperscalers will remain central to AI infrastructure in the near term. They control GPU supply, offer integrated platforms, and are trusted by enterprises. Yet dominance does not imply permanence.

As inference matures, it stops being a feature that differentiates products and becomes a line item that compounds month after month. At that point, enterprises begin to optimize for cost predictability, margin, and operational control.

Deloitte also highlights mounting pressure on data-center capacity and energy availability as AI workloads continue to scale. As inference becomes embedded in live products, reliance on a small number of centralized regions and providers increasingly represents an operational and strategic risk, not just a cost consideration.

Enterprises are therefore incentivized to diversify where inference executes, improving resilience and reducing exposure to regional outages, capacity bottlenecks, and energy constraints.

At the same time, expanding effective GPU supply does not require waiting for new hyperscale data-center builds. Underutilized hardware already exists across enterprises, regional providers, and edge environments.

When inference execution can be coordinated reliably across this fragmented supply, capacity becomes elastic and responsive to demand rather than constrained by centralized provisioning cycles.

Together, these pressures reinforce the case for a second execution layer alongside hyperscale cloud. Hyperscalers remain the natural home for frontier models, burst demand, and compliance-heavy workloads. Coordinated execution layers, by contrast, are structurally suited to absorb steady-state, high-volume inference where cost predictability, locality, and reliability dominate.

The outcome is not displacement, but bifurcation, inference traffic increasingly splits across complementary layers. Historically, this pattern signals that an infrastructure market is entering its next phase, with new winners emerging alongside incumbents rather than in opposition to them.

4. Evaluation Criteria for Web2-Scale Decentralized Inference

To evaluate which decentralized compute platforms can credibly service Web2-scale inference demand, we assess leading networks, Akash, Bittensor, io.net, and Gensyn, against a small set of core infrastructure characteristics.

These criteria represent the minimum technical foundation required to support reproducible, production-grade inference, as opposed to experimental or crypto-native workloads. From a market perspective, the question is not which networks are decentralized, but which can reliably support enterprise inference at scale.

- **Deterministic execution** the ability to produce consistent outputs from identical inputs across machines and environments, enabling reproducibility, auditability, and the safe deployment of agent-based systems.

- **Inference-native runtime execution models** optimized for always-on, low-latency inference rather than batch-oriented or training-first workloads.

- **Abstraction** over hardware heterogeneity shielding developers from GPU variability and node-operator differences to ensure predictable behavior and performance.

- **Production-grade isolation** and security robust sandboxing and clear trust boundaries required to run proprietary models and sensitive data in distributed environments.

Together, these characteristics provide a disciplined framework for distinguishing platforms capable of scaling into the Web2 inference market from those best suited to narrower or experimental use cases.

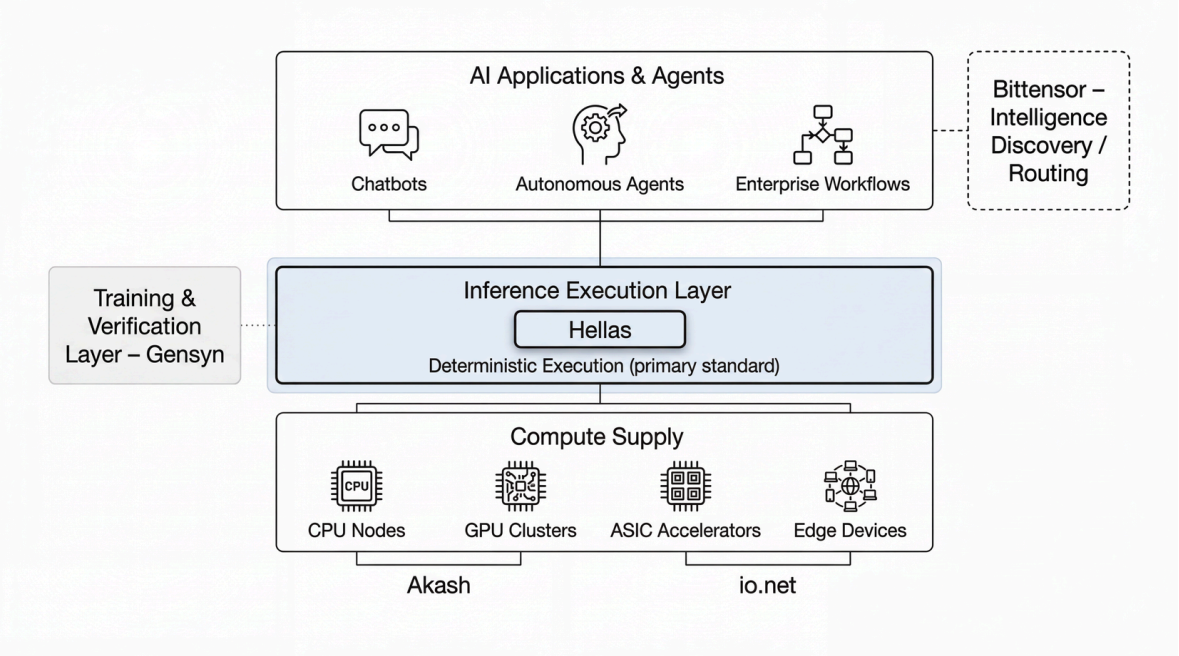
5. Disclaimer and Scope of Analysis

This report does not evaluate decentralized compute platforms solely on advertised compute capacity. While public dashboards provide useful context, capacity alone is an incomplete proxy for production inference readiness.

For example, Akash Network currently reports roughly 500 GPUs, 15.8K CPU cores, and 55 active providers on mainnet, with approximately $4.7M USD in cumulative network spend. io.net reports roughly 3,000 registered GPUs/CPUs and substantial inference volume processed through its API layer. These figures confirm that decentralized compute supply exists and is growing.

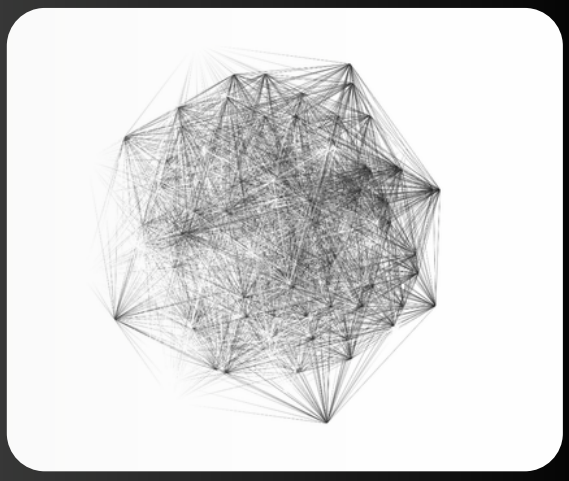
However, such metrics do not answer the questions that determine whether inference can move into production, whether identical requests produce consistent outputs, whether execution is predictable and replayable across heterogeneous nodes, and whether costs scale with utilization rather than reserved capacity. GPU counts aggregate highly variable resources and do not encode execution behavior, determinism, or trust guarantees.

This analysis therefore prioritizes execution-layer characteristics over supply-side abundance. In an inference-dominated market, the binding constraint is not access to compute, but the ability to coordinate reliable, verifiable execution across fragmented infrastructure. Once inference demand becomes predictable and monetizable, supply follows elastically. Accordingly, platforms are evaluated on their structural ability to absorb sustained, always-on inference traffic as it materializes, rather than on headline capacity metrics.



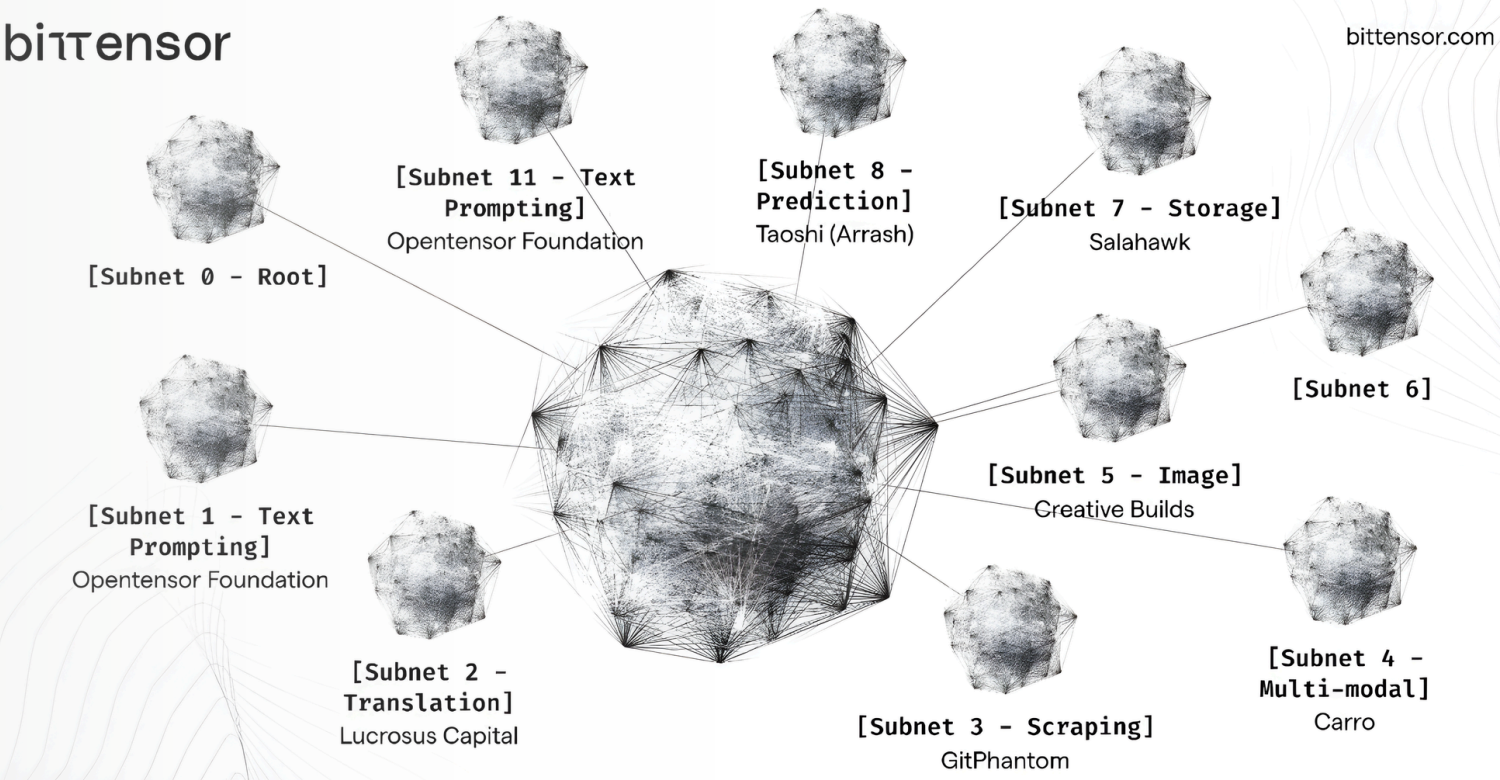
The Emerging AI Execution Stack

This diagram illustrates the separation between intelligence discovery, inference execution, and raw compute supply as AI systems shift toward always-on, economically continuous inference. Hyperscale cloud remains dominant for training and burst workloads, while deterministic execution layers coordinate steady-state inference across heterogeneous and decentralized hardware.



1

bittensor



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

Market intent and architectural orientation

Bittensor is a decentralized intelligence marketplace rather than an inference infrastructure. Its core objective is to incentivize the discovery of high-quality model outputs through open competition, not to standardize execution across a distributed compute base.

The network is organized into subnets, which are application-specific markets that define a task, an evaluation process, and a reward mechanism. Each subnet functions as an independent competition for intelligence, not as a standardized execution environment. This design choice is central to Bittensor's value proposition and explains both its strengths and its limitations when evaluated against Web2 inference requirements.

Deterministic execution and coordination limits

At the protocol and SDK level, Bittensor does not enforce deterministic or reproducible execution. Inference requests are handled by user-defined subnet logic, and outputs are evaluated comparatively rather than verified against a fixed execution baseline. In other words, the system rewards responses that outperform peers, not responses that behave identically across runs, machines, or environments.

This is intentional. Deterministic execution would constrain the competitive dynamics that drive Bittensor's intelligence discovery process. While individual subnets could theoretically impose their own determinism constraints, this property is neither standardized nor enforced at the infrastructure layer.

As a result, reproducibility, auditability, and replayability remain external concerns, managed if at all by application-specific logic rather than by the network itself.

For Web2 inference workloads, particularly agent-based systems where consistent behavior under identical conditions is a prerequisite, this represents a structural misalignment.

The infrastructure challenge shifts from discovering the best answer to coordinating predictable execution, a problem Bittensor is not designed to solve.

Execution model and inference semantics

Bittensor exposes an always-on request/response interface through its Axon and Dendrite components. Axons are the server-side interfaces through which nodes expose their models, while Dendrites are the client-side interfaces used to send requests and collect responses from the network.

Requests are issued by validators, which query multiple miners (nodes running models) and compare their outputs. Validators then allocate rewards based on relative performance. This competitive evaluation loop sits at the core of Bittensor's execution model.

While this architecture supports continuous querying, it should not be conflated with a production-grade inference runtime. Latency, tail behavior, failure handling, and predictability are secondary to incentive alignment. There are no protocol-level service guarantees, and inference exists primarily as an input to a scoring mechanism rather than as a managed, SLA-oriented service.

This model is well-suited to experimentation, benchmarking, and open-ended intelligence discovery, but it diverges from the operational requirements of Web2 inference, where inference is embedded directly into live products, workflows, and user-facing systems.

Hardware heterogeneity as an economic signal

Rather than abstracting away hardware differences, Bittensor explicitly incorporates them into its economic model. Performance variability across GPUs and system configurations is not normalized by a unified runtime or compiler target, instead, it directly influences competitive outcomes. Miners are rewarded for producing superior outputs, and hardware capability is one of several factors that can confer advantage.

From a research and incentive-design perspective, this exposure is coherent. From a Web2 infrastructure perspective, it introduces variability that enterprises typically seek to eliminate. Production inference assumes consistent behavior regardless of where execution occurs, in Bittensor, that consistency is neither guaranteed nor prioritized, reinforcing its orientation toward competition rather than coordinated execution.

Security, trust, and enterprise constraints

The Bittensor SDK includes mechanisms for authenticity and integrity, such as signature verification, body-hash checks, and replay protection. These measures ensure that messages are genuine and unaltered, but they do not provide confidentiality or execution isolation.

Inference requests are routed to third-party miners over standard HTTP, with no native support for confidential execution environments, secure enclaves, or strong tenant isolation. As a result, proprietary models, sensitive prompts, and regulated data require trust assumptions that fall outside the platform's design scope.

This security posture aligns with Bittensor's open, permissionless ethos, but it materially limits applicability for Web2 inference workloads, where data protection, compliance, and isolation are non-negotiable requirements.

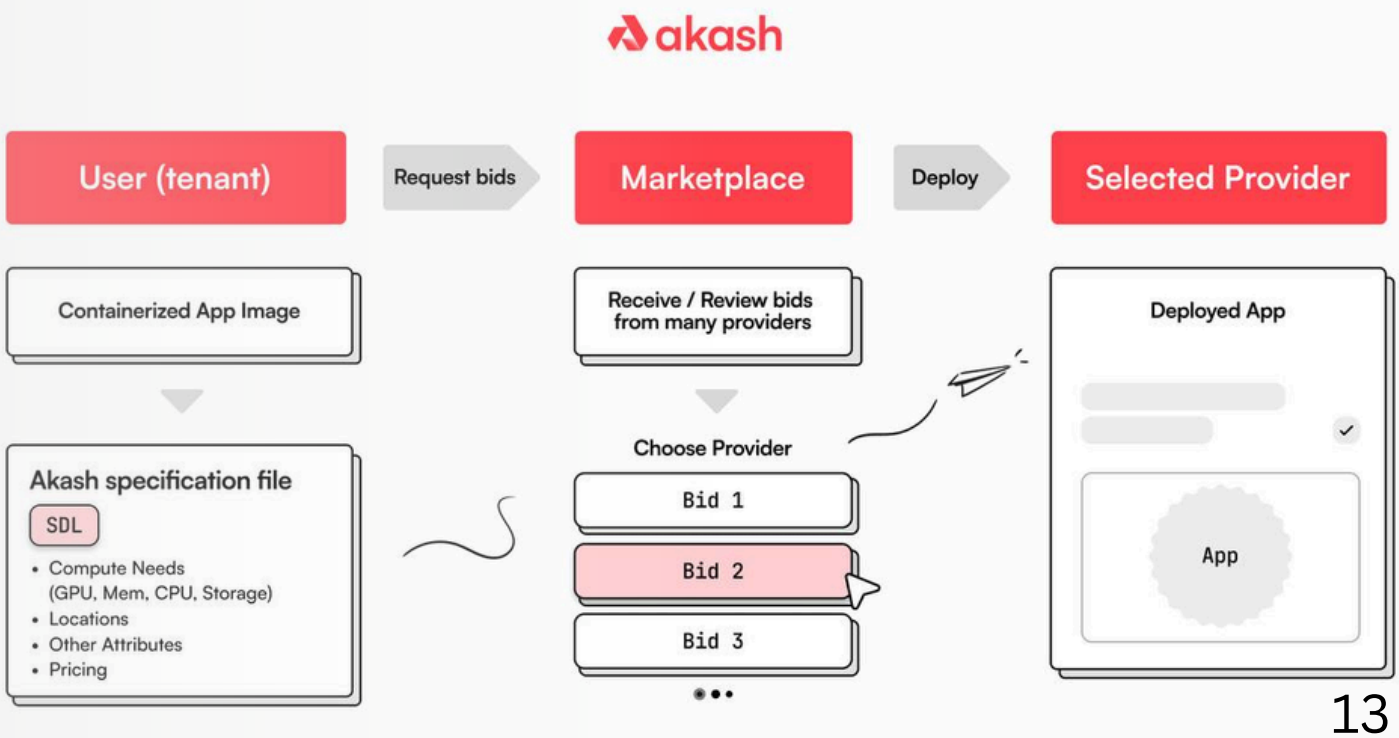
Synthesis: where Bittensor fits

Taken together, these characteristics position Bittensor as a system optimized for decentralized intelligence discovery, not for coordinating production-grade inference across heterogeneous environments. Its architecture excels at surfacing high-quality outputs through competition and aligning incentives in open networks.

In the broader decentralized compute landscape, Bittensor should therefore be understood as complementary rather than substitutive, a discovery and incentive layer that sits adjacent to, but does not replace, execution fabrics explicitly engineered for deterministic, reproducible, and operationally reliable inference.

2

akash


akash
User (tenant)
Request bids
Marketplace
Deploy
Selected Provider
Containerized App Image
Akash specification file
SDL
• Compute Needs (GPU, Mem, CPU, Storage)
• Locations
• Other Attributes
• Pricing
Receive / Review bids from many providers
Choose Provider
Bid 1
Bid 2
Bid 3
Deployed App
App

Market intent and architectural orientation

Akash is best understood as a decentralized cloud infrastructure marketplace rather than an inference execution platform. Its core objective is to surface idle compute supply through price discovery, allowing users to deploy containerized workloads at lower cost than traditional hyperscale cloud providers. The network operates through a reverse auction model.

Users submit deployment requests, while providers (operators running physical or virtual infrastructure) bid to host those workloads. Deployments are described using Akash's Stack Definition Language (SDL), a declarative specification that defines container images, resource requirements, and hardware constraints. This architecture positions Akash as a decentralized alternative to cloud infrastructure provisioning, not as a system for coordinating inference execution semantics.

Deterministic execution and coordination limits

At the platform level, Akash does not provide or enforce deterministic or reproducible execution. Determinism is implicitly delegated to user-supplied container images and the underlying Kubernetes runtime operated by individual providers. While containerization improves environmental consistency, it does not eliminate deeper sources of nondeterminism arising from heterogeneous GPUs, numerical libraries, or scheduling behavior across nodes.

As a result, identical inference requests may yield different outputs depending on where and how they are executed. Reproducibility, auditability, and replayability therefore remain application-layer responsibilities rather than shared infrastructure guarantees.
For Web2 inference workloads, particularly agent-based systems and regulated decision pipelines, this creates a coordination gap.

Once inference becomes always-on and economically continuous, predictable execution behavior is no longer optional. Akash's infrastructure surfaces supply efficiently, but it does not standardize execution across that supply.

Execution model and inference semantics

Akash's execution model is inherited directly from cloud-native container orchestration. Workloads are deployed as long-running containers scheduled on provider-managed Kubernetes clusters. Availability, scaling behavior, health checks, and request handling are managed entirely within the application itself.

This model works well for batch jobs, stateless services, and general-purpose cloud workloads. However, it lacks inference-native semantics such as request-aware scheduling, model residency guarantees, batching, or latency-oriented execution paths. Inference can run on Akash, but only insofar as it is packaged as a generic service, with no platform-level awareness of inference behavior or economics.

As inference traffic scales, this distinction becomes material. Web2 inference systems require execution environments optimized for always-on availability, predictable response times, and graceful degradation under load. Akash provides infrastructure primitives, not execution coordination.

Hardware heterogeneity as an economic signal

Akash explicitly exposes hardware heterogeneity rather than abstracting it away. Deployment manifests must specify GPU vendor, model, and resource attributes, and providers bid accordingly. This gives users fine-grained control over cost and hardware selection, but it also means that execution behavior and performance are tightly coupled to the chosen provider.

There is no unified runtime or compiler layer that normalizes behavior across different GPU architectures. From a Web2 perspective, this shifts operational complexity onto the user. Teams must either restrict deployments to narrow hardware profiles, reducing available supply, or accept variability in inference latency and behavior across environments.

Production inference platforms typically move in the opposite direction, hiding infrastructure differences to deliver consistent execution semantics. Akash's design reflects its roots as a decentralized cloud marketplace rather than as an inference execution layer.

Security, trust, and enterprise constraints

Akash provides standard container-level isolation and authentication mechanisms consistent with Kubernetes-based platforms, including namespace separation and mutual TLS between tenants and providers. These measures establish authenticity and basic access control, but they do not extend to confidential execution.

Akash does not natively support trusted execution environments, secure enclaves, or hardware-backed confidentiality for protecting proprietary models, prompts, or outputs. Instead, trust is mediated through provider selection, audited attributes, and application-level safeguards.

This security posture is coherent for a decentralized infrastructure marketplace, but it limits Akash's applicability for enterprise inference workloads where data sensitivity, regulatory compliance, and execution isolation are core requirements rather than optional enhancements.

Synthesis: where Akash fits

Taken together, Akash is best positioned as a decentralized infrastructure substrate that excels at surfacing and monetizing idle compute supply. Its strengths lie in price discovery, flexible deployment, and access to globally distributed resources.

However, as inference becomes the dominant AI workload, the limiting factor shifts from access to compute to coordination of execution. Web2 inference demands predictable behavior, reproducibility, and verifiability across heterogeneous and untrusted environments.

These properties are not encoded in Akash's execution model. In the emerging inference-dominated market, Akash should therefore be understood as a foundational supply layer rather than an execution layer. It can host inference workloads, but it does not solve the execution coordination problem that determines whether inference traffic can scale reliably and profitably.

Platforms that standardize execution semantics sit upstream of Akash's supply and capture the structural value as inference demand materializes.

3

io.net

Containers

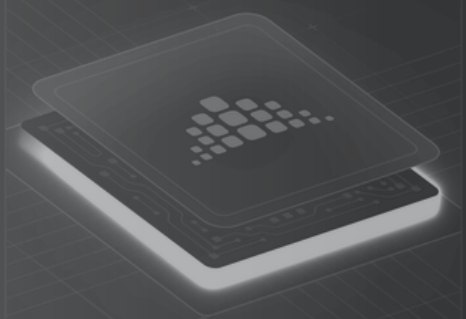
Containers

One-Line Deployment For ML Workloads With Pre-Configured Environments

Virtual Machines

Virtual Machines

AI-Ready VMs in less than 5 Minutes With Pre-Configured ML Frameworks



Ray Cluster

Ray Cluster

Native Support For Containerized AI Workloads With Familiar Tooling

Market intent and architectural orientation

Io.net is a GPU aggregation and provisioning network rather than an inference execution platform. Its primary objective is to unlock underutilized GPU supply by onboarding heterogeneous hardware into a shared marketplace, allowing compute buyers to access capacity that sits outside of traditional hyperscale cloud providers.

The publicly visible components of io.net focus on onboarding workers, machines running GPUs, into the network via setup scripts and launcher binaries. These workers register device-level information and make GPU resources available for allocation. This architecture positions io.net as a supply-side coordination layer, optimized for aggregating raw compute rather than for standardizing how inference executes on that compute.

Deterministic execution and coordination limits

Based on publicly available artifacts, io.net does not expose or document platform-level guarantees around deterministic or reproducible execution. The visible components setup scripts and launcher binaries are concerned with provisioning GPU hardware and initializing worker containers, not with controlling numerical determinism, seed management, or replayability across machines.

Any reproducibility properties therefore appear to be delegated to containerized applications and the underlying GPU hardware rather than enforced by the platform itself. This is consistent with a compute aggregation model, where the platform's role is to surface capacity rather than to coordinate execution semantics.

For Web2 inference workloads, particularly agentic systems or audit-sensitive pipelines, this creates a gap. Once inference becomes always-on and economically continuous, consistent behavior under identical conditions becomes a prerequisite. Io.net surfaces supply efficiently, but it does not impose a shared execution standard across that supply.

Execution model and inference semantics

The public io.net repositories do not reveal an inference-native runtime oriented around serving semantics. Instead, they emphasize GPU worker registration, device configuration, and container lifecycle management. There is no evidence of inference-specific primitives such as request routing, batching, model residency guarantees, or latency-aware scheduling.

Inference workloads can likely be deployed on top of provisioned GPUs, but execution behavior, availability, and performance characteristics are handled outside the platform's visible abstractions. As with traditional infrastructure marketplaces, inference is treated as a workload that runs on compute, not as a first-class execution primitive.

This distinction becomes material as inference traffic scales. Web2 inference systems require always-on availability, predictable response times, and stable execution behavior under variable load. Io.net provides access to GPUs, but it does not coordinate inference execution in a way that supports these requirements natively.

Hardware heterogeneity as an economic signal

Io.net's tooling explicitly exposes hardware heterogeneity. Setup scripts detect GPU vendor, model, CUDA version, and device configuration, and launcher binaries require device-level identifiers. Rather than abstracting execution across heterogeneous hardware, the system registers and manages GPUs as discrete resources with distinct characteristics.

This design maximizes supply inclusion and allows a broad range of hardware to participate, which is advantageous for capacity expansion. However, it also implies that inference performance and behavior vary with provider equipment. For Web2 inference buyers accustomed to abstracted compute units and predictable execution semantics, this shifts the burden of managing variability to the application or orchestration layer.

Security, trust, and enterprise constraints

From the available repositories, io.net relies on standard Docker-based containerization for isolation. There is no visible support for confidential computing, trusted execution environments, or hardware-backed isolation mechanisms. Public artifacts do not document encryption guarantees, secrets management, or protections for proprietary models and prompts beyond what container boundaries provide.

This security posture is typical for early-stage compute aggregation networks, but it constrains adoption for enterprise inference workloads where confidentiality, compliance, and execution isolation are central requirements. Without documented guarantees at the orchestration or execution layers, trust assumptions remain external to the platform itself.

Synthesis: where Akash fits

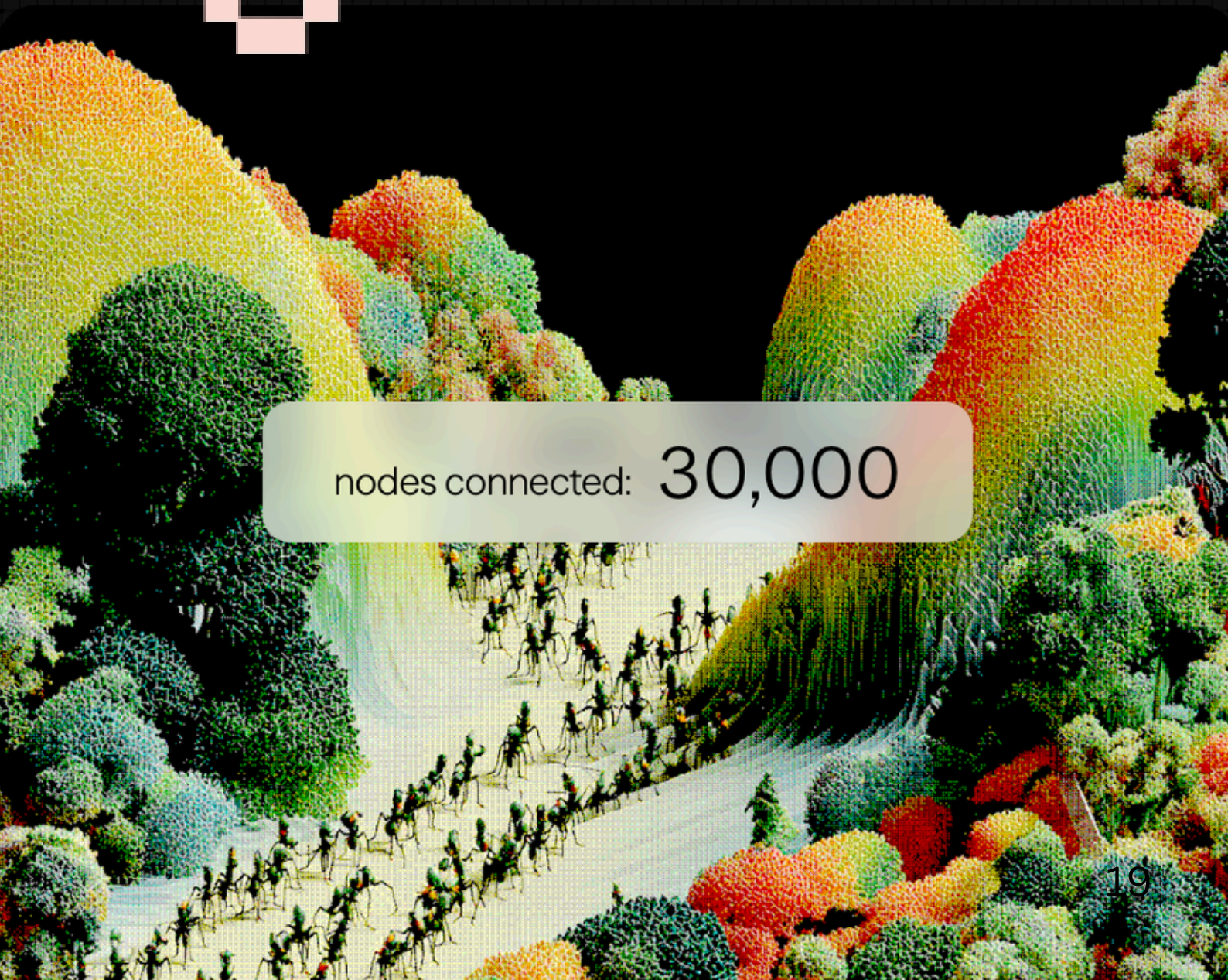
Taken as a whole, io.net occupies an intermediate position within the decentralized compute landscape. Unlike Bittensor, which optimizes for intelligence discovery through incentive-driven competition, io.net is oriented toward aggregating and provisioning raw GPU supply. At the same time, it stops short of the generalized cloud substrate model embodied by Akash, which exposes infrastructure primitives through a fully legible marketplace.

Io.net's public artifacts emphasize hardware onboarding and worker management, while leaving execution semantics, determinism, inference-aware scheduling, and security guarantees, either opaque or externalized. This makes io.net a potentially valuable supply-side layer, but not a standalone inference platform for Web2 workloads that require predictable execution and verifiable behavior.

In an inference-dominated market, io.net functions less as an inference system of record and more as a bridge between fragmented GPU supply and higher-level execution frameworks. The platforms that capture durable value are those that sit above this layer, standardizing execution so that aggregated supply can be reliably monetized at scale.

4

gensyn



Market intent and architectural orientation

Gensyn places a strong emphasis on reproducibility within distributed machine learning workflows. Through its Reproducible Operators (RepOps) framework, the project demonstrates that bitwise-identical execution of specific machine learning workloads across heterogeneous hardware is technically achievable under controlled conditions.

RepOps enforces a fixed ordering of floating-point operations, mitigating nondeterminism arising from hardware variation and numerical instability, and producing identical outputs across CPUs and a broad range of GPU architectures.

However, this capability currently exists as a library and demonstration rather than as a universally enforced property of the protocol. While RepOps signals deep technical sophistication and a first-principles understanding of the reproducibility problem, deterministic execution is not yet established as a mandatory, protocol-level guarantee for all workloads.

For Web2 inference use cases, this distinction is material the capability exists, but its enforcement as a shared execution standard remains optional rather than systemic.

Deterministic execution and coordination limits

Despite its strength in reproducible computation, Gensyn is not architected as an inference-native execution layer. Publicly available code and documentation center on distributed training, particularly reinforcement learning, rather than on inference serving. Core components focus on coordinating peers, distributing gradients, synchronizing models, and verifying work performed across untrusted nodes.

While trained models may generate outputs during evaluation phases, this behavior is incidental to the training workflow rather than indicative of a serving-oriented runtime. There is no evidence of inference-specific primitives such as request routing, batching, latency optimization, or always-on availability guarantees.

As a result, Gensyn's execution model aligns with collaborative training and verification workflows, not with the operational demands of production inference embedded in live applications.

Execution model and inference semantics

Gensyn does not fully abstract away hardware heterogeneity. Instead, it incorporates hardware awareness directly into its execution model. Participants are assigned to different model pools based on GPU capability, and supported hardware targets are explicitly enumerated. RepOps itself requires specifying supported architectures in order to guarantee reproducibility.

This approach enables efficient utilization of diverse hardware, but it also means that execution behavior and model assignment depend on underlying device characteristics. For Web2 inference workloads, where operators typically expect uniform behavior across a fleet, this exposure complicates capacity planning and performance guarantees.

Gensyn prioritizes compatibility and correctness across devices rather than hiding hardware differences behind a single abstraction.

Hardware heterogeneity and fleet-level implications

From an enterprise inference perspective, Gensyn does not currently provide production-grade isolation or confidentiality guarantees. Its security model emphasizes trustless verification, ensuring that work has been performed correctly, rather than protecting the contents of that work from participants in the network.

Training coordination occurs over peer-to-peer communication, with no documented use of trusted execution environments, secure enclaves, or hardware-backed confidentiality mechanisms.

This design choice is coherent with Gensyn's mission to make machine learning computation verifiable and permissionless, but it limits applicability for enterprise scenarios involving proprietary models, sensitive inputs, or regulated data. Confidentiality remains an application-level or external concern rather than a native platform property.

Security, trust, and enterprise constraints

Gensyn's security model is designed primarily for correctness verification, not confidentiality. The protocol's core promise is that distributed participants can be incentivized and checked so that training work is performed as claimed, even across untrusted nodes. This is a different security objective than production inference, where enterprises typically require strict protections for proprietary models, private data, prompts, and outputs.

From the publicly available documentation, Gensyn does not yet present a clear, production-grade story for tenant isolation or confidential execution. There is no standardized requirement for trusted execution environments (TEEs), secure enclaves, or hardware-backed confidentiality guarantees that would prevent other network participants from learning sensitive artifacts.

In practice, this means that enterprise deployments would require additional controls outside the protocol, such as encryption, trusted hardware assumptions, or permissioned participation.

This constraint matters for two reasons. First, many regulated or high-value enterprise workloads cannot tolerate exposing training data or model weights to third-party operators, even if correctness is verifiable. Second, Gensyn's training-first design increases the surface area of sensitive material in motion, gradients, checkpoints, intermediate states, and coordination messages may all carry information that enterprises treat as confidential.

Without strong default confidentiality guarantees, the natural adoption path skews toward open or non-sensitive training workloads, research contexts, and crypto-native experimentation, rather than regulated Web2 environments.

As a result, Gensyn's trust posture is coherent for a permissionless verification protocol, but it remains structurally misaligned with enterprise requirements for confidentiality, compliance, and isolation, especially relative to inference execution layers that aim to minimize trust assumptions through deterministic replayability, standardized runtimes, and security primitives designed for production serving.

Synthesis: where Gensyn fits

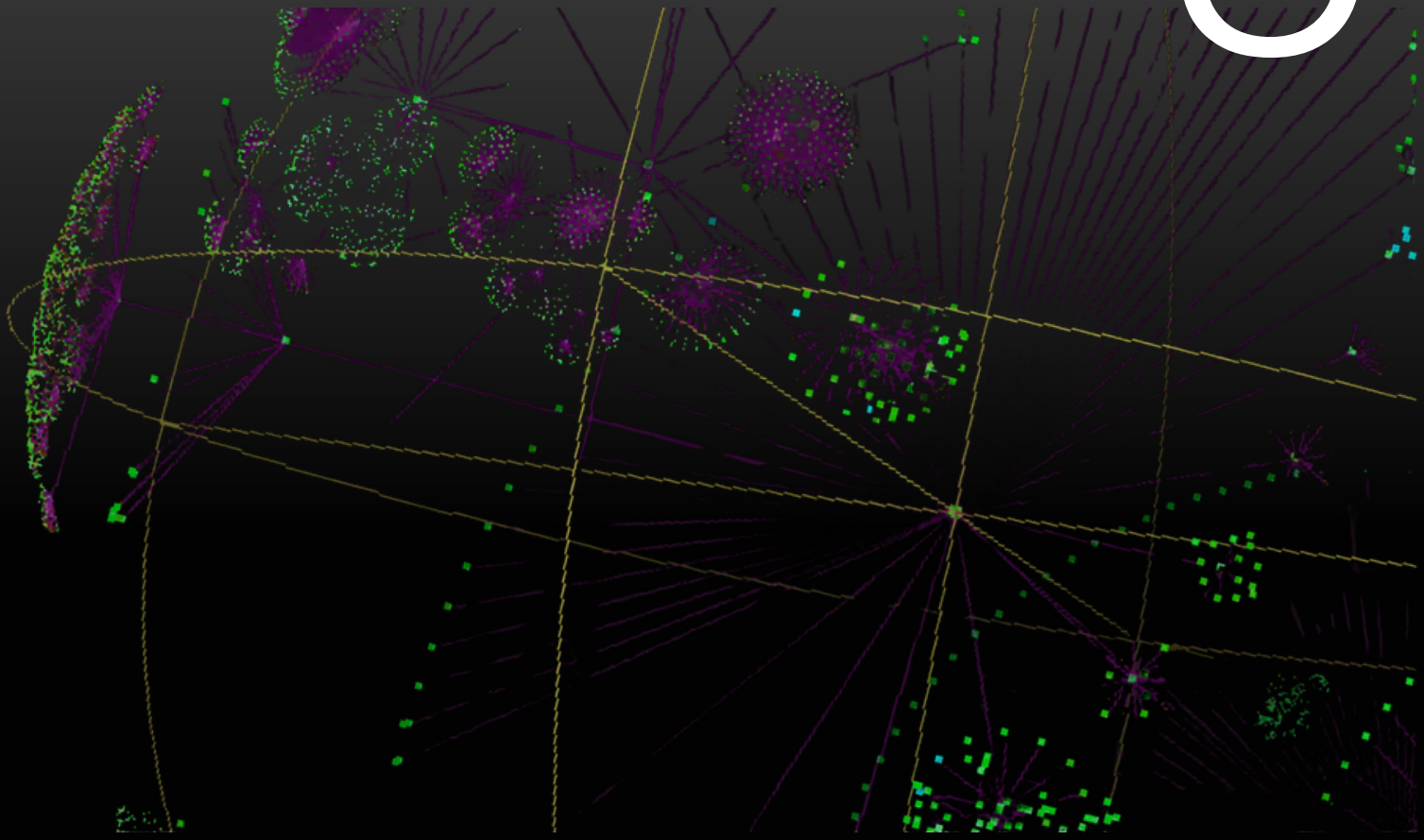
Taken together, Gensyn is best understood as a protocol for verifiable, distributed machine learning training rather than as a decentralized inference platform. Its architecture is deeply informed by the challenges of reproducibility, correctness, and coordination across untrusted hardware, and it offers some of the most advanced tooling in the ecosystem for addressing nondeterminism in machine learning computation.

However, these strengths are applied primarily to training workflows, not to inference serving. Gensyn demonstrates what deterministic execution could look like, but it does not yet package that capability into an inference-native runtime suitable for Web2 deployment.

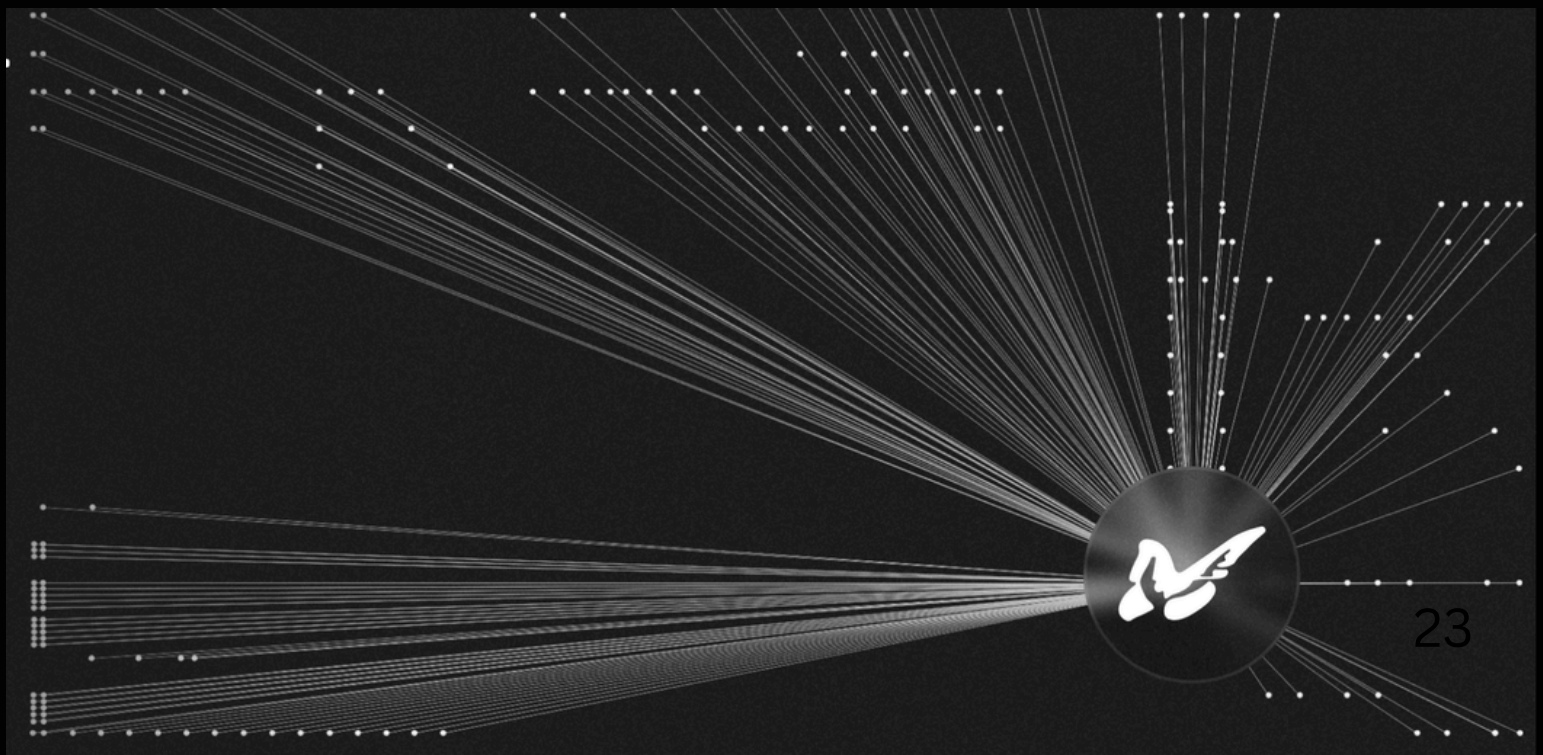
Notably, current implementations are exercised within a testnet and research-oriented environment rather than a production mainnet, reinforcing Gensyn's positioning as a training-first protocol.

Within the decentralized compute landscape, Gensyn occupies a supporting role, advancing the foundations of verifiable ML, while the challenge of operational, always-on inference execution across fragmented infrastructure remains an open frontier.

5



PRIME Intellect



Market intent and architectural orientation

Prime Intellect is a training-oriented distributed machine learning platform, not as an inference execution layer. Its primary focus is enabling large-scale reinforcement learning through coordinated training, rollout generation, and verifier-based evaluation, rather than standardizing how inference executes in production environments.

The platform combines three core components, prime-rl for large-scale RL training, verifiers for environment definition and evaluation, and prime-cli for GPU resource discovery and orchestration. Inference exists within this architecture primarily as a supporting mechanism for training rollouts, not as a first-class, always-on serving primitive.

From a Web2 infrastructure perspective, this positions Prime Intellect closer to a research-driven training stack than to a production inference platform embedded into live products, workflows, or user-facing systems.

Abstraction over hardware heterogeneity

Prime Intellect does not provide deterministic execution guarantees for inference. Inference within prime-rl is handled via a vLLM backend, inheriting vLLM's default behavior, which prioritizes throughput and batching efficiency over reproducibility.

There is no protocol-level enforcement of deterministic sampling, seed management, or bitwise reproducibility across runs, machines, or environments. Any determinism properties are therefore incidental and workload-specific rather than systemic.

For Web2 inference workloads, particularly agentic systems or regulated pipelines where reproducibility, auditability, and replayability are mandatory, this represents a structural limitation. Execution correctness may be acceptable for training rollouts, but it is not standardized as a shared execution guarantee across the platform.

Hardware heterogeneity and fleet-level implications

Prime Intellect explicitly exposes hardware heterogeneity rather than abstracting it away. The prime-cli tooling requires users to select GPU types, memory configurations, vCPU counts, and data-center locations when provisioning resources.

This design maximizes transparency and control for training workloads, where hardware selection and optimization are part of the experimentation process. However, it also means that execution behavior, performance, and cost depend directly on underlying device characteristics.

For Web2 inference workloads, where developers expect uniform behavior across a fleet regardless of hardware, this exposure introduces operational complexity. Hardware variability is managed at the orchestration layer rather than normalized by a unified execution standard.

Security, trust, and enterprise constraints

Prime Intellect provides basic authentication and sandboxing mechanisms suitable for development and research workflows, but it does not expose production-grade guarantees for confidentiality or tenant isolation when running on third-party infrastructure.

There is no documented support for trusted execution environments, hardware-backed confidentiality, encrypted model execution, or isolation guarantees that would protect proprietary models, prompts, or outputs. Security is oriented toward correct execution of training workflows, not toward enterprise-grade inference isolation.

As a result, enterprises deploying sensitive or regulated inference workloads would need to rely on external controls and trust assumptions beyond what the platform natively provides.

Synthesis: where Prime Intellect fits

Taken together, Prime Intellect is best positioned as a highly capable distributed reinforcement learning platform, optimized for scaling training, coordinating rollouts, and evaluating agent behavior across large GPU fleets.

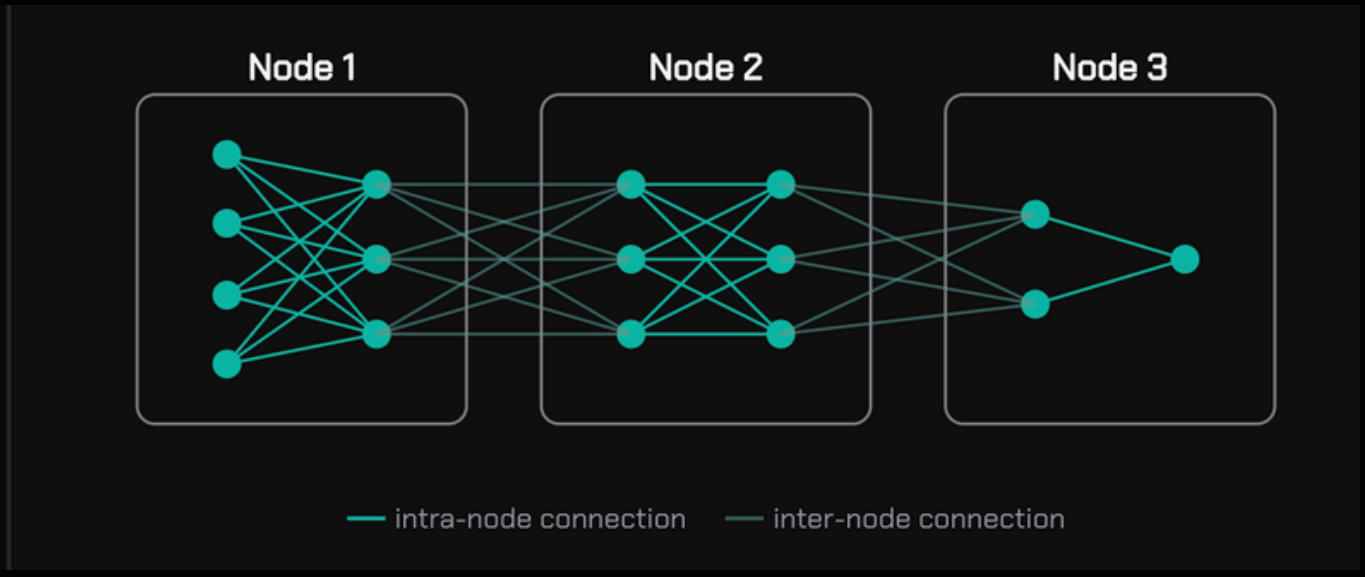
Its architecture demonstrates strong engineering depth in training orchestration, but it does not encode the execution guarantees required for Web2-scale production inference. Determinism, inference-native semantics, hardware abstraction, and enterprise-grade isolation remain outside the platform's core design goals.

Within the broader decentralized compute landscape, Prime Intellect should therefore be understood as supportive rather than competitive with inference execution layers. It advances the state of distributed training, while the challenge of standardized, deterministic, always-on inference execution remains a separate layer in the stack.

PLURALIS RESEARCH

Node Ø

Node0-7.5B is a permissionless, multi-participant, model-parallel pretraining run occurring over the Internet. Anyone with a single 16GB+ GPU can join. Node0 allows participants to collaboratively train a model much larger than any individual could train alone.



Market intent and architectural orientation

Pluralis Research is best understood as a decentralized, collaborative training initiative rather than an inference execution platform. Its core systems, node0 and AsyncPP, are designed to enable distributed model pretraining and to optimize training efficiency across heterogeneous, volunteer-operated hardware.

Node0 focuses on enabling participants to jointly train large models by partitioning computation graphs across peers and coordinating gradient updates over the network. AsyncPP complements this by improving asynchronous pipeline parallelism during training, reducing idle time and improving throughput. Both systems are explicitly oriented toward training scalability and participation rather than inference serving.

From a Web2 inference perspective, this distinction is decisive. Pluralis Research is not attempting to standardize inference execution, expose inference endpoints, or coordinate always-on model serving. Its architectural intent is collaborative learning, not production inference infrastructure.

Deterministic execution and coordination limits

Pluralis Research systems do not provide deterministic execution guarantees for inference. While node0 includes manual seed setting to improve training reproducibility, the codebase explicitly acknowledges that execution order and data assignment are not guaranteed during runtime.

This level of nondeterminism is acceptable, and often expected, in distributed training environments, where approximate reproducibility is sufficient and performance trade-offs dominate. However, no mechanisms exist to enforce deterministic inference behavior, replayability, or identical outputs across nodes or runs.

As a result, Pluralis Research systems cannot support inference workloads that require consistent behavior under identical inputs. From an execution standpoint, they verify participation in training rather than enforcing predictable runtime semantics.

Execution model and inference semantics

Pluralis Research does not expose an inference runtime model. Node0 operates as a training coordinator, running distributed training loops, averaging gradients, and synchronizing model state across peers. AsyncPP is a training optimization technique, not a serving framework.

There are no inference endpoints, request-routing primitives, batching mechanisms, or always-on serving abstractions present in the repositories. Inference, if performed at all, is incidental to training evaluation rather than a first-class execution mode.

For Web2 inference adoption, this represents a fundamental gap. Supporting production inference would require a complete architectural shift away from training-centric coordination toward serving-oriented execution semantics.

Hardware heterogeneity and abstraction

Pluralis Research systems explicitly expose hardware heterogeneity rather than abstracting it away. Node0 supports single-GPU execution and requires explicit device selection, with separate code paths for CPU, CUDA, and MPS environments.

While the system is accessible, supporting consumer-grade GPUs with relatively low memory requirements, it does not normalize performance or behavior across hardware types. There is no abstraction layer that would allow inference or execution to behave uniformly across a heterogeneous fleet.

This design lowers the barrier to participation for training, but it shifts operational complexity onto users and prevents predictable execution behavior, a trade-off that is incompatible with Web2 inference expectations.

Security, trust, and enterprise constraints

Pluralis Research provides basic authorization and integrity checks, such as token-based authentication and code integrity verification. However, it does not provide production-grade isolation or confidentiality guarantees. There is no use of trusted execution environments, secure enclaves, encrypted execution, or hardware-backed confidentiality mechanisms.

Model parameters, training data, and intermediate states may be visible to participating nodes, consistent with an open, collaborative training model.

For enterprise inference workloads, where proprietary models, private data, and regulatory constraints are common, this security posture is insufficient. Confidentiality remains an external concern rather than a native platform property.

Synthesis: where PluralisResearch fits

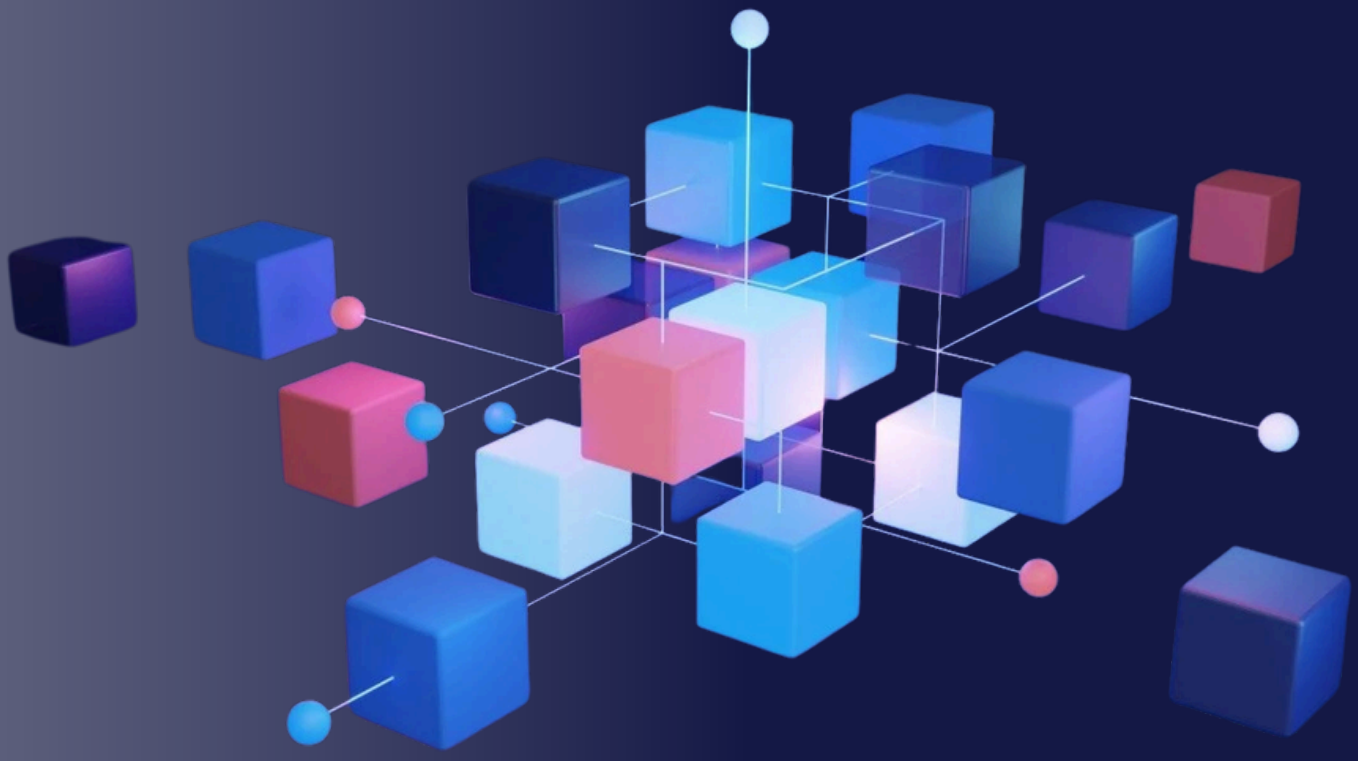
Taken together, Pluralis Research is best understood as a decentralized collaborative training framework rather than an inference infrastructure platform. Its architecture is optimized for participation, scalability, and experimentation in distributed model training, not for serving inference in production environments.

While the project demonstrates meaningful innovation in coordinating training across heterogeneous, internet-scale nodes, these strengths do not translate to inference execution. Determinism, inference semantics, hardware abstraction, and enterprise-grade isolation are outside its design scope.

Within the decentralized compute landscape, Pluralis Research occupies a complementary role, advancing open and participatory model training, while the challenge of standardized, deterministic, always-on inference execution remains unaddressed.

7

Cocoon



Market intent and architectural orientation

Cocoon is best understood as a decentralized inference serving network optimized for confidential execution rather than deterministic coordination. Its architecture centers on running AI models inside trusted execution environments (TEEs), with the goal of allowing third-party GPU operators to serve inference workloads while minimizing trust in the operator.

The platform prioritizes cryptographic verification of what code is running and where it is running, using Intel TDX attestation and TON-based settlement. This positions Cocoon closer to a privacy-preserving inference hosting layer than to an execution standardization layer.

Deterministic execution and coordination limits

Cocoon does not enforce deterministic or reproducible inference execution at the platform level. Determinism is delegated to the underlying model runtime and hardware rather than being enforced as a shared execution property.

Trusted execution environments guarantee isolation and code integrity, but they do not constrain numerical nondeterminism, runtime scheduling differences, or hardware-specific behavior. As a result, identical inputs may yield different outputs across runs or nodes.

For Web2 inference systems require replayability, auditability, or consistent agent behavior, while Cocoon solely verifies execution identity rather than execution semantics.

Execution model and inference semantics

Cocoon is designed for always-on, low-latency inference serving with streaming responses, rather than for batch processing or training workflows. Requests are routed through proxies to TDX-protected workers and settled via off-chain payment channels optimized for frequent, small inference calls.

While this architecture aligns with real-time inference patterns, it lacks inference-native coordination primitives such as request-aware scheduling, latency guarantees, or standardized service-level behavior. Performance variability is handled through reputation mechanisms rather than execution guarantees.

Inference is therefore treated as a hosted service rather than as a standardized execution artifact.

Hardware heterogeneity and abstraction

Cocoon does not abstract away hardware heterogeneity. GPU models, firmware configuration, and performance characteristics are explicit and economically relevant within the system.

Workers must meet strict vendor-specific requirements, including confidential computing support and manual configuration, and pricing reflects performance differences across devices. Hardware variability is exposed rather than normalized.

This design is coherent for a TEE-based network but introduces operational complexity that Web2 inference platforms typically seek to hide behind a uniform execution surface.

Security, trust, and enterprise constraints

Cocoon's strongest differentiation lies in confidentiality and isolation. Intel TDX and remote attestation provide strong guarantees that inference executes inside verified environments and that prompts and responses remain private from operators.

However, these guarantees focus on code identity and confidentiality rather than deterministic correctness. Model integrity is verified via hashing, but execution behavior is not independently verifiable or replayable. For enterprises, Cocoon reduces operator trust assumptions but does not eliminate reliance on probabilistic runtime behavior.

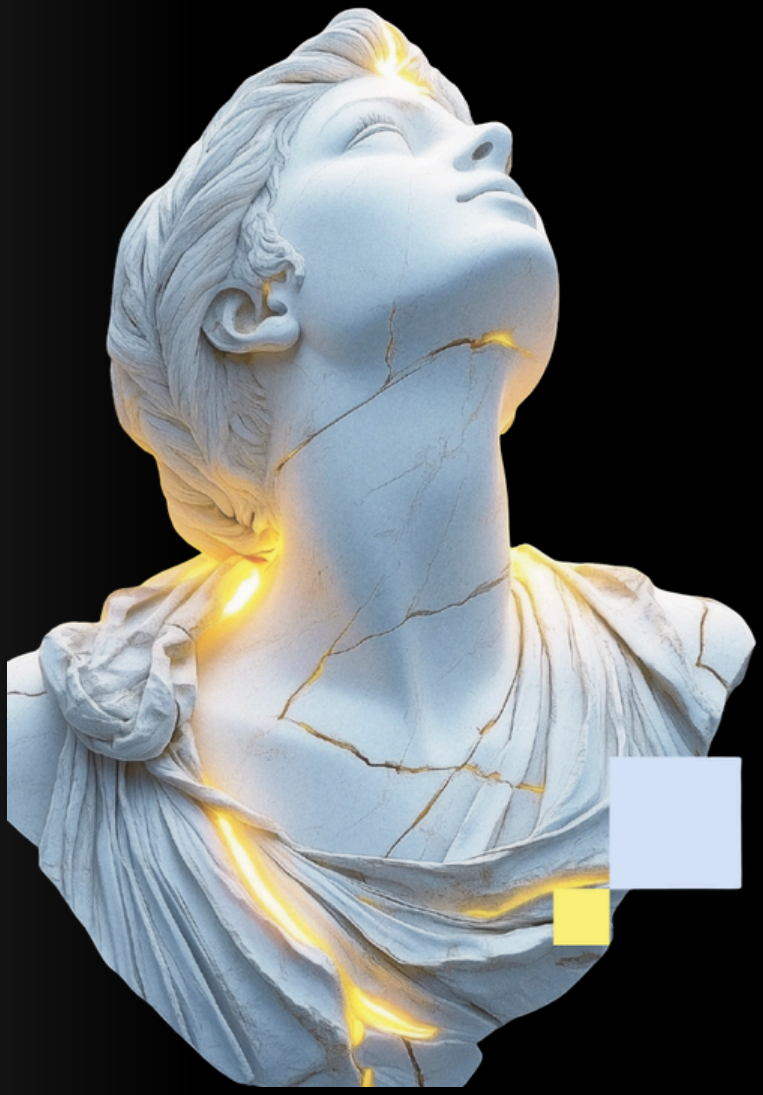
Synthesis: where Cocoon fits

Taken together, Cocoon is best understood as a decentralized, privacy-first inference serving network rather than a deterministic execution layer. Its architecture excels at confidential inference and trust minimization across untrusted operators.

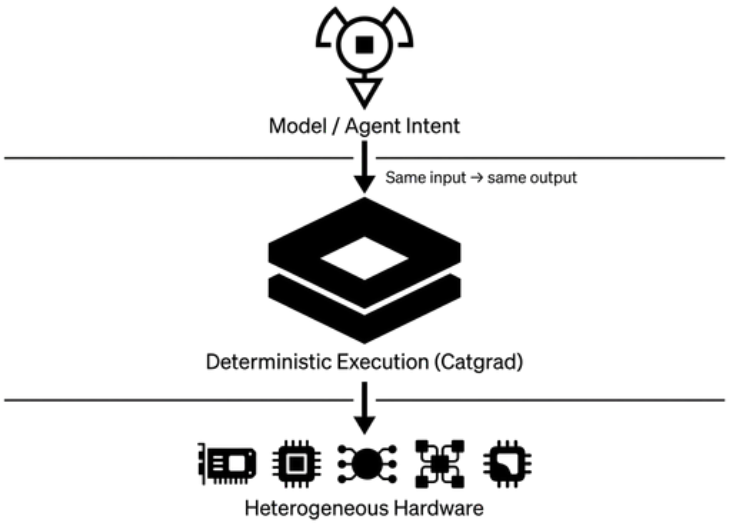
However, Cocoon does not standardize inference execution semantics across heterogeneous environments. Determinism, reproducibility, and behavioral coordination remain outside the platform's guarantees.

Within the decentralized compute landscape, Cocoon occupies a supporting role, addressing confidentiality and verifiability, while the challenge of standardized, reproducible inference execution at Web2 scale remains unresolved.

8



HELLAS



Deterministic execution

Hellas is architected around deterministic execution as priority rather than an optional application-level concern. Unlike most decentralized compute networks, which delegate reproducibility to containers, user code, or individual operators, Hellas enforces determinism at the model-execution layer through Catgrad. Catgrad is a compiler that transforms code into fixed, symbolic tensor programs.

These programs remove runtime graph construction, floating-point non-associativity, and framework-level nondeterminism, producing consistent outputs for identical inputs across machines and environments. This design directly addresses a structural blocker for Web2 inference adoption.

Enterprises cannot operationalize large-scale inference pipelines if outputs vary across nodes, regions, or execution contexts. Hellas reframes inference from best-effort execution into auditable computation, enabling reproducibility, regression testing, compliance review, and agent safety guarantees that probabilistic runtimes cannot reliably provide.

Critically, deterministic execution also enables predictable cost modeling and cacheability, aligning inference economics with utilization rather than reservation. This property is foundational for steady-state inference workloads, where cost predictability matters more than peak throughput. Hellas is structurally aligned with deterministic inference, a prerequisite for scaling agentic systems, regulated AI workloads, and verifiable compute markets.

At the same time, deterministic execution introduces a higher bar for tooling, compiler maturity, and developer experience. Hellas' advantage is architectural rather than incremental, but its adoption will depend on how effectively these guarantees can be delivered without creating new workflow friction.

Inference-native execution model

Hellas is allows inference-first by construction, not retrofitted from training workflows or generic container orchestration. Where Akash and io.net expose Kubernetes-style deployment primitives, and Gensyn optimizes for distributed training, Hellas treats inference as a portable execution artifact. Models are compiled into small, self-contained programs that require no Python runtime, no heavyweight frameworks, and no environment-specific assumptions.

This enables always-on inference without the operational overhead typical of cloud-based serving stacks. Providers execute compiled artifacts directly and return results that can be verified without full re-execution. The execution model is therefore compatible with bursty demand, low-latency responses, and high request volumes the defining characteristics of inference embedded in products, agents, and user-facing systems.

Abstraction over hardware heterogeneity

Hellas abstracts hardware heterogeneity at the execution layer rather than exposing it to developers or pushing complexity upstream. In most decentralized networks, GPU variability leaks into application logic, users must specify hardware constraints, accept performance variance, or design around heterogeneous execution. Hellas avoids this by compiling models into deterministic tensor programs that execute consistently across supported backends.

Hardware differences still affect performance, but not correctness or behavior. This separation mirrors how traditional software abstracts CPUs while allowing competition on performance underneath. Providers compete on speed, cost, and reliability, while users interact with a uniform execution surface.

This abstraction is essential for Web2 adoption, where predictable behavior is a baseline requirement and hardware selection cannot be an application-level concern. The trade-off is that Hellas must continuously expand and optimize its supported backends to ensure that abstraction does not become a bottleneck to supply inclusion or performance.

Production-grade trust, verification, and security

Hellas prioritizes verifiability over blind trust, aligning with enterprise and regulated-market requirements. Rather than assuming trusted operators or relying solely on container isolation, Hellas leverages deterministic execution to make results objectively checkable. Providers commit to outputs that can be disputed, audited, or cryptographically verified without reproducing the full computation.

This enables fraud detection, SLA enforcement, and future integration with zero-knowledge proofs or dispute-resolution mechanisms. Hellas's core insight is through determinism computation is reproducible, and correctness becomes verifiable rather than assumed.

That said, confidentiality guarantees and enterprise-grade isolation will remain an important complement for certain workloads. Hellas' architecture reduces trust requirements but does not eliminate the need for layered security as it moves toward broader enterprise adoption.

Market positioning and scale trajectory

Hellas is not a generic compute marketplace. It is an execution and coordination layer that unlocks the economic potential of decentralized compute. Akash monetizes idle infrastructure. Io.net aggregates GPU supply.

Bittensor incentivizes probabilistic intelligence discovery. Gensyn verifies distributed training. Hellas sits orthogonally to all four, providing the missing execution fabric that allows distributed hardware to serve production inference reliably.

Synthesis: where Hellas fits

As inference becomes the dominant mode of AI deployment, the limiting factor is no longer access to compute, but the ability to execute models predictably, repeatedly, and at scale across fragmented and heterogeneous infrastructure. Web2 inference workloads fail not because GPUs are scarce, but because probabilistic execution, opaque runtimes, and cloud-style economics break down under continuous use.

Hellas is designed around this constraint. By treating deterministic execution as the unit of coordination rather than raw compute supply. Hellas enables distributed hardware to behave like a coherent inference substrate. Models become portable, auditable programs, execution becomes verifiable rather than assumed, and cost scales with utilization rather than reservation.

From an infrastructure perspective, Hellas represents a shift from renting compute to standardizing execution. That shift aligns directly with where AI is going, away from experimentation and toward persistent, economically meaningful inference. The remaining challenge is execution, adoption, and ecosystem build-out, not conceptual fit.

Next Crucial Questions to Answer

If this architectural shift is real, why hasn't it already become standard?

What breaks when this moves from theory into production?

Where does adoption slow, and why?

Which users feel the pain first, and which never will?

What has to be true for this to work at scale?

Sources

Macro & Industry Context
AI Infrastructure & Inference Economics

- Deloitte – Tech Trends 2026: AI Infrastructure & Compute Strategy
- https://www.deloitte.com/us/en/insights/topics/technology-management/tech-trends/2026/ai-infrastructure-compute-strategy.html

Academic / Technical Foundations

- Reproducible Operators / Deterministic ML (arXiv)
- https://arxiv.org/abs/2512.04051
- Supporting technical paper (PDF)
- https://cdn.sanity.io/files/2bt0j8lu/production/95f2e8619ec05dcdaf834f9457830b3ea2b81824.pdf

Decentralized Compute Networks (Dashboards & Public Data)

- Akash Network – Network Statistics https://stats.akash.network/
- io.net – Platform Overview https://io.net/
- Gensyn – Network Dashboard https://dashboard.gensyn.ai/

Competitor GitHub Repositories

Bittensor

- Core protocol https://github.com/opentensor/bittensor

Akash Network

- Node implementation https://github.com/akash-network/node
- Provider software https://github.com/akash-network/provider
- Documentation https://github.com/akash-network/docs

Sources

Io.net

- Organization repositories https://github.com/ionet-official
- Launch binaries https://github.com/ionet-official/io_launch_binaries
- Setup scripts https://github.com/ionet-official/io-net-official-setup-script

Gensyn

- RL Swarm (training coordination) https://github.com/gensyn-ai/RL-Swarm
- Reproducible Operators (RepOps demo) https://github.com/gensyn-ai/repops-demo

Pluralis Research

- Node implementation https://github.com/PluralisResearch/node0
- AsyncPP (protocol research) https://github.com/PluralisResearch/AsyncPP

Prime Intellect

- PRIME-RL (RL training framework) https://github.com/PrimeIntellect-ai/prime-rl
- PRIME CLI (GPU orchestration) https://github.com/PrimeIntellect-ai/prime-cli
- Verifiers (RL environments & evaluation) https://github.com/PrimeIntellect-ai/verifiers

TON Cocoon (Telegram)

- Cocoon core repository https://github.com/TelegramMessenger/cocoon
- Cocoon smart contracts https://github.com/TelegramMessenger/cocoon-contract

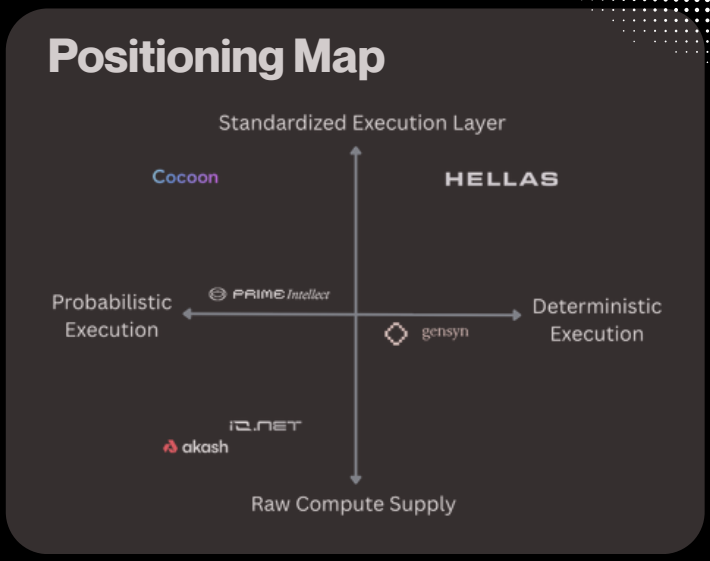
Hellas AI

- Catgrad deterministic compiler https://github.com/hellas-ai/catgrad

Crypto + AI Thesis

Deloitte's Tech Trends 2026 projects that inference will drive roughly two-thirds of AI compute as models become always-on. Hyperscalers are optimized for shared, bursty workloads, but inference forces over-provisioning, idle GPU capacity, and rising operating costs.

This creates a new infrastructure market. The major players fall into three categories: compute marketplaces, training networks, and execution layers, each addressing different parts of this demand.



bittensor

Bittensor operates an intelligence discovery market, using economic incentives to surface high-quality model outputs. It is designed for experimentation and benchmarking rather than operational reliability.

Execution is probabilistic and non-reproducible, thus Bittensor captures value in discovery markets, not in production inference infrastructure.

gensyn

Gensyn focuses on verifiable and reproducible distributed training, proving deterministic computation is technically achievable.

However, it is training-first rather than inference-native and does not offer a standardized serving runtime. Its addressable market is distributed training verification, not production inference execution.

akash

Akash is a decentralized cloud marketplace that monetizes idle compute through price discovery. It provisions infrastructure but does not control how AI inference executes once deployed.

As a result, it serves cost-sensitive infrastructure buyers, not enterprises seeking predictable inference economics or production-grade reliability.

PRIME Intellect

Prime Intellect is a distributed training and reinforcement-learning platform where inference is used to support training workflows.

Execution remains probabilistic and hardware-dependent. As a result, Prime Intellect captures value in training orchestration and experimentation, not in enterprise inference infrastructure.

io.net

io.net aggregates underutilized GPUs into a shared marketplace, expanding access to raw compute capacity. Its focus is supply onboarding and orchestration rather than inference execution.

Without standardized runtime behavior, io.net captures value at the compute aggregation layer, not in servicing persistent inference demand.

PR
PLURALIS RESEARCH

Pluralis enables collaborative, participatory model training across decentralized contributors. Its systems are designed for coordination and inclusion, not for standardized inference serving.

Without deterministic execution guarantees, Pluralis addresses research and training collaboration markets rather than production inference demand.

Cocoon

Cocoon provides confidential inference using trusted execution environments, reducing operator trust requirements. While it secures where inference runs, it does not standardize execution behavior or outputs. Cocoon therefore captures value as a privacy and trust layer, not as deterministic inference infrastructure.

HELLAS

Hellas enables deterministic inference by standardising how AI models run across heterogeneous hardware. Through its compiler, Catgrad, GPU providers execute verifiable compiled artifacts directly.Among its immediate applications is production-grade inference infrastructure, where deterministic execution delivers immediate value.