Computing-Aware Traffic Steering
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

H. Wang
Q. Li
Pengcheng Laboratory
Y. Jiang
Tsinghua Shenzhen International Graduate School & Pengcheng Laboratory
M. Xu
Tsinghua University

2 March 2026

An Open, Decentralized, and Scalable Framework for Large Language Model Inference
draft-wang-cats-odsi-00

Abstract

Large Language Model (LLM) inference is increasingly deployed as a networked service, yet existing deployments rely primarily on centralized infrastructure and trusted operators. Such designs limit openness, concentrate resource ownership, and constrain scalability to the capacity of individual providers. At the same time, LLM inference introduces execution characteristics (e.g., strict sequential dependencies, large intermediate activations, and tight latency requirements) that are not well supported by existing network, transport, or coordination mechanisms in open environments.

This document specifies an open, decentralized, and scalable framework for executing LLM inference across independently operated and mutually untrusted participants. The framework treats inference as a distributed, layer-wise execution process subject to explicit deadlines, rather than as a monolithic computation or best-effort service. It combines layer-aware activation transport and routing, decentralized coordination among heterogeneous compute resources, and security mechanisms that provide accountability and correctness without assuming trusted execution.

This document focuses on the architectural framework, design rationale, problem definition, challenges, and solution space of the Open, Decentralized, and Scalable Inference framework (ODSI). It does not specify concrete wire protocols, message formats, or protocol state machines. Such protocol-level specifications are to be defined in separate documents that build upon the framework described herein.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://kongyanye.github.io/draft-wang-cats-odsi/draft-wang-cats-odsi.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wang-cats-odsi/.

Discussion of this document takes place on the Computing-Aware Traffic Steering Working Group mailing list (mailto:cats@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/cats/. Subscribe at https://www.ietf.org/mailman/listinfo/cats/.

Source for this draft and an issue tracker can be found at https://github.com/kongyanye/draft-wang-cats-odsi.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 September 2026.

Copyright Notice

Table of Contents

1. Introduction

Large Language Models (LLMs) have become a foundational component of modern networked applications, supporting tasks such as natural language understanding, code generation, and interactive assistants. Inference for these models is typically delivered as an online service, where user requests are transmitted to remote servers and processed incrementally to generate output tokens. Unlike traditional network services or offline model training workloads, LLM inference exhibits strict sequential dependencies, maintains per-request execution state, and imposes tight latency constraints on model response.

Today, most large-scale LLM inference deployments rely on centralized infrastructure operated by a small number of providers. Centralization simplifies coordination, scheduling, and state management, but it also concentrates control, limits participation, and couples scalability to the capacity, geography, and policies of individual operators. As model sizes and inference demand continue to grow, these structural limitations motivate the exploration of alternative execution paradigms that can support broader participation and more elastic scaling.

This document introduces the Open, Decentralized, and Scalable Inference framework (ODSI), an architectural framework for executing LLM inference across independently operated and heterogeneous compute resources. ODSI considers environments in which participants are mutually untrusted and connected only through the public Internet. The framework focuses on how inference execution can be coordinated, secured, and scaled under such conditions, without assuming centralized control or trusted execution environments.

1.1. Motivation for Open and Decentralized Inference

Centralized inference architectures assume that compute resources, network paths, and execution environments are under common administrative control. While these assumptions enable tight optimization and simplified coordination, they impose structural constraints that become increasingly pronounced as model sizes continue to grow. Modern LLMs require substantial compute throughput

and memory capacity, and deploying a single model instance often exceeds the capabilities of an individual server. As a result, inference deployments increasingly rely on multiple servers to host and execute a single model, raising operational complexity and cost.

At the same time, a large volume of distributed compute resources remains underutilized. Many devices and servers possess limited memory capacity or modest compute performance, preventing them from hosting complete model instances despite having available compute resources. These constraints lead to fragmented and wasted capacity, particularly at the network edge or within smaller organizations, where individual nodes cannot independently support large models even though aggregate resources may be sufficient.

Centralized cloud-based inference also introduces data movement and privacy concerns. User inputs must be transmitted to remote data centers for processing, increasing exposure to data leakage and raising privacy risks in sensitive applications. In addition, aggregating large volumes of inference traffic at centralized endpoints places sustained pressure on network bandwidth, increases transmission and queuing delays, and creates service bottlenecks that limit horizontal scalability. As demand grows, scaling inference capacity requires proportional expansion of centralized infrastructure and network provisioning, which may not be economically or operationally sustainable.

An open and decentralized inference model seeks to address these limitations by allowing independently operated participants to contribute partial compute resources without requiring prior trust relationships or centralized admission. By distributing inference execution across many nodes, including resource-constrained and edge devices, this paradigm enables inference capacity to scale elastically with participation. Placing computation closer to data sources can also reduce data movement, mitigate bandwidth bottlenecks, and improve responsiveness in certain deployment scenarios.

However, decentralization fundamentally changes the inference execution environment. Participants may vary widely in compute performance, memory capacity, network connectivity, and availability, and some participants may behave maliciously or rationally rather than altruistically. Moreover, LLM inference is inherently stateful, i.e., the generation of each output token depends on all previous tokens, commonly represented through cached intermediate values such as key寰砺alue (KV) caches. These characteristics make inference sensitive to delays, failures, and inconsistencies, and prevent it from being treated as a stateless or best-effort distributed task.

ODSI is motivated by the need to support open participation and elastic scaling while preserving the correctness, timeliness, and reliability required for practical LLM inference. The framework addresses how inference can be executed across decentralized and heterogeneous resources while remaining usable as an interactive network service.

1.2. Scope and Non-Goals

This document defines the architectural framework, problem formulation, and design considerations for decentralized LLM inference under open participation. It identifies the key challenges introduced by decentralization, including state management, latency constraints, heterogeneity, and adversarial behavior, and describes the high-level mechanisms used to address these challenges within the ODSI framework.

This document does not specify concrete network protocols, wire formats, message encodings, or protocol state machines. It also does not mandate specific model architectures, execution platforms, hardware accelerators, or economic systems. Where cryptographic, incentive, or coordination mechanisms are discussed, they are described at an abstract level to illustrate design intent rather than to prescribe particular implementations.

Protocol-level specifications, interoperability requirements, and implementation details are to be defined in other documents that build upon the framework presented here.

1.3. Design Principles

The ODSI framework is guided by the following design principles:

* Open Participation: Any independently operated participant may contribute compute resources without requiring centralized admission or prior trust, subject to mechanisms that provide accountability and abuse resistance.

* Decentralized Coordination: Inference execution is coordinated without assuming a single trusted controller, relying instead on distributed mechanisms that tolerate heterogeneity, failures, and adversarial behavior.

* State-Aware Execution: The framework explicitly accounts for the stateful and sequential nature of LLM inference, including the management of intermediate execution state across tokens and layers.

* Deadline Sensitivity: Inference execution is treated as a latency-sensitive process, where intermediate steps are subject to explicit timing constraints rather than best-effort delivery.

* Scalability Through Composition: The framework is designed to scale by composing many independent contributors, allowing overall inference capacity to grow with participation rather than centralized provisioning.

These principles inform the architectural choices and mechanisms described in the remainder of this document.

2. Terminology

This section defines the terminology used throughout this document. Phrases in upper-case refer to other defined terms.

ACTIVATION

Intermediate numerical data produced by executing a model layer during inference. ACTIVATIONS are consumed by subsequent layers and may be transmitted across the network between participants.

CONTROL PLANE

The non-latency-critical path responsible for coordination, verification, and enforcement functions, including cryptographic IDENTITY registration, STAKE management, SLASHING, REWARD settlement, and REPUTATION updates.

DEADLINE

A time constraint by which a specific inference step, such as a layer execution or ACTIVATION delivery, must complete to preserve end-to-end responsiveness.

EXECUTION COMMITMENT

A cryptographically signed declaration by a PARTICIPANT indicating intent to execute a specific inference step under defined inputs and DEADLINES, enabling later verification.

IDENTITY

A persistent, self-generated identifier bound to a PARTICIPANT, typically realized as a public寰蒐rivate key pair.

INFERENCE PLANE

The latency-critical path responsible for performing inference-related work, including LAYER execution, ACTIVATION transport, ROUTING, RE-ROUTING, and failure signaling.

LAYER

A discrete computation stage within an inference pipeline. LAYERS are executed sequentially for each token step and may be assigned to different execution participants.

PARTICIPANT

An independently operated entity that contributes compute, memory, or network resources to ODSI and participates using a cryptographic IDENTITY.

REPUTATION

A persistent performance signal associated with a PARTICIPANT, derived from historical correctness, DEADLINE adherence, availability, and throughput. REPUTATION influences task assignment and reward rates.

RE-ROUTING

The process of dynamically changing the ROUTING or LAYER assignment of an ongoing inference request in response to failures, performance degradation, or DEADLINE pressure.

REWARD

An economic payment issued to a PARTICIPANT for successfully completing an assigned inference task.

ROUTING

The selection of execution participants and network paths for ACTIVATION transport and LAYER execution, informed by DEADLINES and observed performance.

SLACK

The difference between an allocated DEADLINE budget and the expected execution time for an inference step. SLACK represents tolerance to variability and delay.

SLASHING

An enforced economic penalty applied to a PARTICIPANT when verifiable misbehavior is detected, such as incorrect output, missed DEADLINES, or commitment violations.

STAKE

A quantity of economic value locked by a PARTICIPANT as collateral for participation. STAKE enables accountability, economic deterrence, and Sybil resistance

3. Problem Definition

This section defines the core problem addressed by the ODSI framework. The goal is to formalize the execution constraints and failure modes that arise when LLM inference is performed across open, distributed, and independently operated compute resources. These constraints collectively define what it means for decentralized inference to be correct, timely, and usable.

3.1. Layer-Dependent Execution Pattern

LLM inference executes as a fixed sequence of model layers applied repeatedly for each generated token. The output of each layer constitutes an intermediate activation that is required as input to the next layer in the sequence. Execution therefore forms a strict dependency chain at layer granularity.

In a decentralized setting, different layers may be executed on different nodes. This introduces an explicit requirement that intermediate activations be transferred between nodes in the correct order and without duplication or omission. Any execution framework must preserve layer ordering and ensure that each layer operates on the correct input corresponding to a specific inference request and token position.

3.2. Stateful Token-by-Token Progression

Inference proceeds incrementally, generating tokens one at a time. Each token depends on an execution state accumulated from all previous tokens, commonly including cached intermediate values such as KV caches. This state is logically persistent over the lifetime of the inference request.

As a result, inference requests exhibit execution affinity, i.e., successive tokens must either be processed by nodes that already possess the relevant state or incur the cost of state transfer or reconstruction. Failures, delays, or inconsistencies in state handling directly affect correctness and latency. The problem therefore includes maintaining coherent per-request state across a sequence of distributed execution steps.

3.3. Activation Delivery with Timing Constraints

For interactive applications, inference must progress under tight latency constraints. Each layer execution contributes both computation delay and communication delay, and delays in earlier layers propagate to all subsequent layers within the same token generation step.

This creates implicit per-layer timing constraints, i.e., intermediate activations must be delivered within bounded time to sustain acceptable end-to-end response latency. The problem is not merely reliable delivery, but timely delivery under variable network conditions and heterogeneous execution speeds.

The activation delivery problem is defined as follows. Given a sequence of layer-dependent computations distributed across multiple nodes, how can intermediate activations be delivered and processed in order, within time bounds, despite variability in network latency, compute throughput, and node availability?

3.4. Open Participation and Adversarial Behavior

ODSI assumes an open execution environment in which nodes may join or leave without centralized admission and are operated by independent parties. Participants may differ significantly in performance, reliability, and incentives, and some may behave maliciously or rationally rather than cooperatively.

The problems to solve therefore includes:

* Detecting incorrect or inconsistent execution results,

* Attributing actions to specific participants,

* Preventing abuse such as equivocation, free-riding, or denial of service,

* Enabling accountability without assuming trusted execution environments.

Any viable solution must address correctness and liveness under these conditions while remaining compatible with open participation.

3.5. Limitations of Existing Execution and Transport Methods

Existing execution and transport methods do not directly address the problem defined above. Best-effort networking does not account for execution dependencies or timing constraints. Traditional distributed computation frameworks assume stable membership, trusted operators, or coarse-grained tasks. Centralized schedulers do not extend naturally to environments without common administrative control.

The problem addressed by ODSI is therefore not solved by simply distributing computation or improving transport performance. It requires a framework that explicitly integrates execution dependencies, state management, timing constraints, and participant accountability into the design.

4. System and Threat Assumptions

This section specifies the assumptions under which the ODSI framework operates. It defines the participating entities, communication and execution conditions, and the classes of failures and adversarial behavior the framework is designed to tolerate. No centralized trust, privileged operators, or trusted execution environments are assumed.

4.1. Participants and Roles

The system consists of a set of independently operated nodes that participate in inference execution. Nodes may assume one or more of the following roles:

* *Clients* initiate inference requests and receive generated outputs. A client may or may not also participate in execution.

* *Execution Nodes* perform inference computation, typically executing one or more model layers for specific inference requests. Execution nodes may differ in compute capacity, memory availability, and supported hardware.

* *Coordination Nodes* participate in control-plane functions such as identity management, stake accounting, verification, and settlement. These roles may be co-located with execution nodes or operated separately.

All participants are identified by persistent cryptographic identities. No global trust relationships are assumed among participants, and no role is restricted to a fixed or privileged set of operators.

4.2. Network and Execution Assumptions

Nodes communicate over the public Internet using unreliable, asynchronous networks. Message delivery may experience variable latency, reordering, duplication, or loss. No assumptions are made about bounded network delay or synchronized clocks, except where explicitly stated by higher-layer mechanisms.

Execution nodes are heterogeneous. They may differ in:

* Compute throughput and supported instruction sets,

* Available memory and storage,

* Network bandwidth and latency,

* Availability and uptime.

Nodes may join or leave the system at arbitrary times. Execution may be interrupted due to failures, preemption, or voluntary withdrawal. The system does not assume trusted execution environments or hardware-based attestation, and correctness cannot be inferred solely from successful message delivery.

4.3. Adversary and Failure Model

The framework assumes the presence of faulty, rational, and malicious participants. Nodes may deviate arbitrarily from prescribed behavior, including but not limited to:

* Returning incorrect or fabricated inference outputs,

* Withholding results or responding after deadlines,

* Equivocating by providing inconsistent results to different peers,

* Attempting to free-ride without performing assigned computation,

* Launching denial-of-service or resource exhaustion attacks.

In addition to malicious behavior, the system must tolerate non-malicious failures such as crashes, network partitions, and transient performance degradation.

The adversary is not assumed to control a majority of system resources globally, but may control multiple identities unless mitigated by Sybil-resistance mechanisms. The framework does not assume confidentiality of intermediate activations unless explicitly provided by higher-layer mechanisms.

Under these assumptions, the framework aims to provide:

* Safety: incorrect inference results can be detected and attributed,

* Liveness: inference can make progress despite failures and churn,

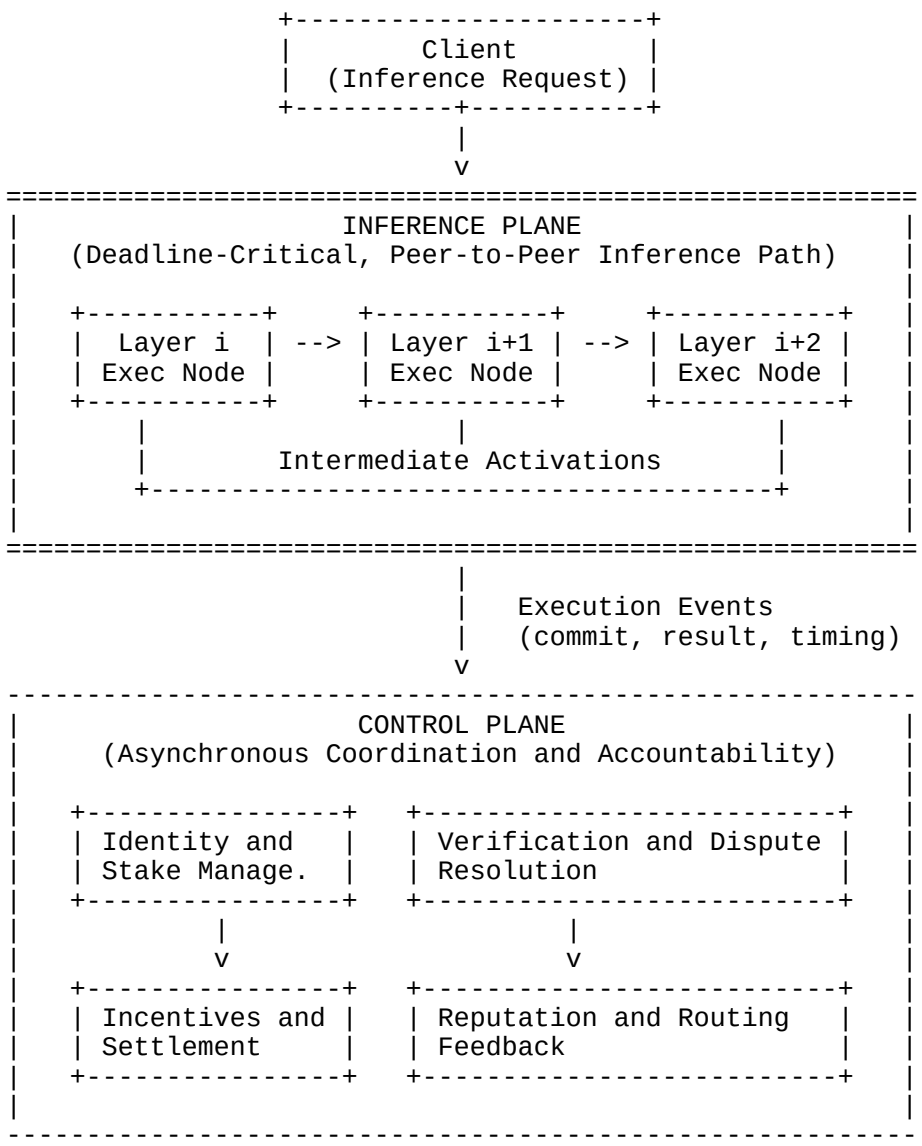
* Accountability: misbehavior can be penalized without relying on trusted authorities.

5. ODSI Framework Overview

This section provides a high-level overview of the ODSI framework. The framework defines how large-scale inference execution can be coordinated across open, heterogeneous, and independently operated resources while preserving correctness, timeliness, and accountability.

ODSI is structured as a layered framework that integrates execution, coordination, and incentive mechanisms. It does not mandate a specific implementation or protocol stack, but instead defines architectural components and their interactions.

5.1. High-Level Architecture



At a high level, ODSI separates inference execution into two logically distinct planes:

* Execution Plane: Responsible for performing inference computation and delivering intermediate results under latency constraints.

* Control Plane: Responsible for identity management, coordination, verification, accounting, and enforcement.

The execution plane operates in a peer-to-peer fashion and is optimized for low-latency, deadline-sensitive computation. The control plane operates asynchronously and does not block inference progress, allowing execution to proceed even in the presence of coordination delays.

This separation enables inference to remain responsive while still supporting accountability and correctness in an open environment.

5.2. Layer-Wise Distributed Execution Model

ODSI adopts a layer-wise execution model in which inference computation is decomposed into sequential stages corresponding to the layers of the inference graph. Each stage may be executed by a different execution node, and intermediate results are transferred between nodes as needed.

Execution proceeds incrementally for each inference request, with explicit association between:

* A specific inference request,

* A specific token generation step,

* A specific computation layer.

This explicit structuring allows the framework to reason about execution dependencies, timing constraints, and correctness at layer granularity. It also enables flexible placement of computation across nodes with varying capabilities, without requiring any single node to host the entire inference workload.

5.3. Open Participation and Resource Contribution

ODSI is designed to support open participation without centralized admission control. Any node may contribute resources to inference execution by assuming execution or coordination roles, subject to framework-defined requirements for identity, accountability, and correctness.

Resource contribution is flexible and may include:

* Compute capacity for executing specific computation stages,

* Memory for maintaining execution state,

* Network capacity for transporting intermediate results.

Nodes are not required to support complete inference execution. Instead, they may contribute partial resources aligned with their capabilities. This allows resource-constrained or edge nodes to participate meaningfully, while enabling aggregate inference capacity to scale with the number of participants.

5.4. Scalability Considerations

Scalability in ODSI is achieved through decentralization, decomposition, and asynchronous coordination. By distributing execution across many independently operated nodes, the framework avoids reliance on centralized bottlenecks.

Key scalability properties include:

* Horizontal scaling: Inference capacity increases with the number of participating nodes.

* Elastic participation: Nodes may join or leave without global reconfiguration.

* Local decision-making: Execution placement and routing decisions can be made using local observations rather than global state.

The framework is designed to tolerate heterogeneity and churn while maintaining bounded coordination overhead. As inference demand grows, scalability is achieved by expanding participation rather than by increasing the capacity of centralized infrastructure.

6. Execution and Coordination Mechanism

This section describes how inference execution is coordinated across distributed participants in the ODSI framework. It focuses on how computation is assigned, how execution state is preserved, how heterogeneous resources are coordinated, and how failures are handled during inference execution.

6.1. Layer Assignment and Path Affinity

Inference execution in ODSI is organized as a sequence of layer executions. For each inference request, layers are assigned to execution nodes based on their capabilities, availability, and observed performance characteristics.

ODSI introduces the notion of execution path affinity, whereby successive layers and token steps of the same inference request preferentially follow a consistent sequence of nodes. Path affinity reduces the need to transfer execution state and intermediate data, thereby improving latency and reducing network overhead.

Layer assignment decisions may be adapted dynamically in response to changing network conditions or node availability. However, reassignment is performed conservatively to avoid excessive state migration or disruption of ongoing inference execution. Under typical operating conditions, stable execution paths are expected to dominate, and recovery-related latency remains within acceptable bounds for interactive inference.

6.2. Handling Stateful Inference and KV Cache

Inference execution maintains per-request state across token generation steps. This state commonly includes cached intermediate values such as KV caches, which are required for efficient generation of subsequent tokens.

ODSI treats execution state as logically associated with an execution path rather than with a single node. State may be:

* Retained locally by execution nodes across successive steps,

* Transferred explicitly when execution is reassigned,

* Reconstructed when transfer is infeasible or too costly.

This document does not mandate a specific state representation or transfer mechanism. Instead, it defines coordination requirements that ensure state consistency and correctness across execution steps.

6.3. Heterogeneous Compute Coordination

Execution nodes in ODSI are heterogeneous in compute performance, memory capacity, and network connectivity. Coordination mechanisms must account for this heterogeneity when assigning computation and routing intermediate results.

Nodes may advertise capabilities and performance metrics, such as execution throughput or observed latency. Coordination decisions may incorporate these metrics to balance load, avoid bottlenecks, and satisfy timing constraints.

The framework supports partial participation, allowing nodes to execute only those computation stages that align with their capabilities. This enables broad participation without requiring uniform hardware or resource provisioning.

6.4. Failure Handling and Recovery

ODSI is designed to tolerate failures during inference execution, including node crashes, network disruptions, and missed execution deadlines.

Failure handling strategies include:

* Detecting stalled or failed execution steps through timeout or absence of expected outputs,

* Reassigning computation to alternative nodes when failures occur,

* Reconstructing execution state when necessary to resume inference.

The framework prioritizes forward progress and bounded recovery cost. Failures may result in degraded performance or recomputation, but should not compromise correctness or global system stability.

Recovery mechanisms are coordinated without assuming centralized control and do not require halting unrelated inference requests.

7. Deadline-Driven Execution and Performance Considerations

Interactive inference services impose strict latency requirements that shape execution, coordination, and resource selection decisions. This section describes how ODSI reasons about deadlines, identifies sources of latency, and manages trade-offs between flexibility and timely execution.

7.1. Deadline Semantics and Slack

In ODSI, inference execution is associated with explicit or implicit deadlines derived from application-level responsiveness requirements. Deadlines apply not only to end-to-end inference requests but also to intermediate execution steps, such as individual layer computations within a token generation cycle.

Each execution step may be assigned a deadline budget, representing the maximum allowable time from input availability to output production. The difference between the allocated budget and the expected execution time is referred to as slack. Slack captures tolerance to variability in computation and communication and serves as a key signal for execution planning.

Slack is consumed as inference progresses. Delays incurred at earlier steps reduce the slack available to downstream steps, making subsequent execution increasingly time-sensitive. The framework therefore prioritizes maintaining positive slack throughout execution to preserve responsiveness.

7.2. Latency Sources and Bottlenecks

End-to-end inference latency arises from multiple sources, including:

* Computation time for executing individual layers,

* Data transfer time for delivering intermediate results,

* Queuing delays caused by contention for compute or network resources,

* Coordination overhead for assignment and verification.

In decentralized environments, these latency components are highly variable and may fluctuate over short time scales. Bottlenecks may shift dynamically due to changes in network conditions, node availability, or workload distribution.

ODSI does not assume that any single latency source dominates. Instead, it treats latency as a composite effect and seeks to minimize the risk of deadline violations by accounting for both computation and communication delays when coordinating execution.

7.3. Routing and Scheduling Considerations

Routing and scheduling decisions in ODSI are informed by deadline constraints and observed performance. Execution steps may be routed through nodes that offer favorable trade-offs between compute throughput, network latency, and reliability.

Scheduling decisions may incorporate:

* Estimated computation time based on historical performance,

* Network round-trip latency and variability,

* Current queue occupancy or load,

* Remaining slack for the inference request.

The framework favors execution paths that preserve slack and reduce the probability of deadline violations. However, routing decisions are made using incomplete and potentially stale information, and must therefore tolerate uncertainty.

7.4. Trade-offs Between Flexibility and Timeliness

ODSI balances execution flexibility against the need for timely completion. Allowing frequent reassignment or dynamic reconfiguration can improve robustness and load balancing but may introduce additional coordination overhead and state transfer costs.

Conversely, favoring stable execution paths improves predictability and reduces overhead but may limit the system宴洯 ability to respond to failures or performance degradation.

The framework does not prescribe a single optimal balance. Instead, it defines a design space in which implementations may tune the degree of flexibility based on workload characteristics, deployment conditions, and performance objectives. The guiding principle is to favor timely execution in the common case, while retaining sufficient adaptability to preserve progress under adverse conditions.

8. Security and Accountability Framework

This section describes the security and accountability mechanisms assumed by the ODSI framework. The framework operates in an open environment with mutually untrusted participants and therefore relies on cryptographic mechanisms and economic accountability rather than trusted operators or centralized enforcement.

8.1. Cryptographic Identity

Each participant in the system is associated with a persistent cryptographic identity, typically represented by a public宴菟rivate key pair [RFC6979]. This identity serves as the basis for authentication, attribution, and accountability across all interactions.

All inference-related actions, including execution commitments, result submissions, and coordination messages, are bound to the participant宴洯 identity through digital signatures. This binding ensures that actions can be reliably attributed to specific participants without relying on centralized identity providers.

Identities are self-generated and do not imply trust. The framework assumes that a single entity may control multiple identities unless constrained by additional mechanisms such as economic bonding or resource-based admission.

8.2. Verifiable Execution Actions

ODSI requires that execution-related actions be verifiable, meaning that they can be independently checked for consistency, correctness, or policy compliance after the fact.

Verifiable actions may include:

* Commitments to execute specific computation stages,

* Submission of intermediate or final execution outputs,

* Timing assertions related to execution deadlines.

Verification does not require continuous oversight during execution. Instead, it relies on cryptographic commitments, hashes, and signed messages that allow third parties to reconstruct and evaluate execution behavior when disputes arise.

This approach enables detection of incorrect execution, equivocation, or deadline violations without imposing synchronous verification on the critical execution path [Byzantine].

8.3. Stake-Based Participation and Accountability

ODSI employs stake-based participation as a foundational mechanism for accountability and Sybil resistance. To serve inference requests, participants are required to lock a quantity of economic value as stake, which acts as collateral against misbehavior.

Stake introduces a real economic cost to participation and creates persistent consequences for incorrect or unreliable behavior. When misbehavior is verified寰敗uch as incorrect execution, commitment violations, or repeated deadline failures寰敗take may be partially or fully forfeited according to predefined rules.

Stake directly enables accountability by ensuring that identities are bound to economic risk. It also provides an economic basis for Sybil resistance: while identities are inexpensive to create, meaningful participation and influence require proportional stake. As a result, large-scale Sybil attacks require substantial capital commitment and expose the adversary to high risk of loss.

The influence and opportunities afforded to a participant may be proportional to both locked stake and historical performance. This ensures that influence reflects sustained contribution rather than identity count alone.

8.4. Accountability Without Trusted Parties

The framework is designed to provide accountability without assuming trusted coordinators, validators, or execution environments. Accountability is achieved by combining identity-bound actions with verifiable evidence and enforceable consequences.

When misbehavior is detected, responsibility can be attributed to specific identities based on signed execution records. Consequences, such as penalties or exclusion from future participation, can then be applied according to defined rules.

This design ensures that participants are held accountable for their actions while preserving open participation and decentralization. Trust is replaced by verification and consequence, allowing the system to operate securely in adversarial environments.

9. Incentives and Economic Considerations

ODSI operates in an open environment where participation is voluntary and participants are assumed to act in their own interest. As a result, correct and timely execution cannot be assumed to arise from cooperation alone. This section outlines the economic mechanisms that align participant incentives with system objectives, ensuring reliable execution under decentralized operation.

9.1. Motivation for Incentive Mechanisms

In a decentralized inference environment, participants contribute compute, memory, and network resources that incur real costs. Without explicit incentives, participants may decline execution assignments, deprioritize inference workloads, or abandon execution paths when conditions become unfavorable.

ODSI therefore associates inference execution with explicit economic rewards [Bitcoin]. Each successfully executed layer earns a payment, creating a direct linkage between contributed work and compensation. Reward levels may vary based on execution conditions, including:

* Tighter execution deadlines, which impose higher performance requirements,

* Placement on latency-critical or bottleneck segments of an execution path,

* Historical reliability and performance of the executing participant.

By rewarding each completed execution unit, ODSI enables fine-grained accounting and encourages participants to contribute resources proportionally to their capabilities.

9.2. Costly Misbehavior and Deterrence

For incentives to be effective, misbehavior must be economically disadvantageous. ODSI assumes that participants may behave strategically, including submitting incorrect activation outputs, violating execution commitments, or missing assigned deadlines.

Misbehavior triggers penalties through the control path. Slashing events may occur in response to:

* Incorrect or invalid activation outputs,

* Mismatches between committed execution inputs and revealed results,

* Failure to meet agreed execution deadlines.

Penalties are calibrated to exceed the expected gains from cheating or shirking, ensuring that rational participants cannot profit from misbehavior even if detection is probabilistic. Slashing may involve forfeiture of locked stake, loss of accrued rewards, or other economically meaningful consequences.

Penalties are applied only when sufficient cryptographic and execution evidence exists to attribute responsibility to a specific identity. This ensures that deterrence is precise and does not require centralized trust or continuous supervision.

9.3. Reputation and Long-Term Participation

While per-layer payments and penalties shape short-term behavior, long-term reliability is reinforced through reputation mechanisms. Reputation reflects a participant寰冭 historical performance across multiple dimensions, including:

* Deadline adherence rate,

* Output correctness and consistency,

* Availability and responsiveness,

* Sustained execution throughput over time.

Reputation directly influences future participation opportunities. Participants with strong reputations may receive higher task volumes, preferential assignment to latency-critical execution paths, higher reward rates, or access to tighter deadlines. Conversely, participants with poor or unstable reputations may receive fewer assignments, reduced compensation, or eventual exclusion.

Reputation complements direct economic incentives by encouraging sustained, honest participation across many execution sessions. Together, per-layer rewards, slashing-based deterrence, and reputation-driven coordination create a self-reinforcing environment in which rational participants are motivated to behave reliably, enabling scalable and decentralized inference execution.

10. Inference Path and Control Path Separation

ODSI separates inference execution into an inference path and a control path in order to reconcile strict latency requirements with the need for security, accountability, and open participation (analogy to Lightning [Lightning]). The inference path is responsible for latency-critical execution and data movement required to generate inference outputs. The control path is responsible for identity management, economic coordination, verification, and enforcement. Together, they form a coherent architecture that decouples performance-sensitive execution from governance and accountability functions.

10.1. Design Rationale

Interactive inference requires execution progress within tight and predictable time bounds. Operations such as cryptographic identity registration, stake management, global verification, or reward settlement cannot be placed directly on the critical execution path without violating these constraints.

At the same time, an open and decentralized environment requires mechanisms to deter misbehavior, attribute responsibility, and enforce incentives. These mechanisms inherently involve coordination, state persistence, and potentially delayed resolution.

The separation between the inference path and the control path addresses this tension by allowing inference execution to proceed optimistically and independently, while ensuring that execution remains observable, attributable, and enforceable. Lightweight

checks on the inference path reduce the likelihood that incorrect results propagate to users, while the control path provides definitive validation, economic settlement, and long-term accountability.

10.2. Inference Path Execution Properties

The inference path carries all latency-critical activities required for inference execution. This includes:

* Layer-wise computation across participating nodes,

* Transport of intermediate activations and execution state,

* Routing and re-routing decisions based on network and execution conditions,

* Failure detection and signaling to enable timely recovery.

The inference path is optimized for low latency, minimal coordination overhead, and predictable progress under Internet variability.

Inference path execution is optimistic but constrained. Participants are not blindly trusted, and incorrect execution is mitigated through the following mechanisms:

* Execution commitments: Participants commit to execution inputs, layer identifiers, and deadlines before revealing outputs, preventing adaptive or inconsistent behavior.

* Selective redundancy: For high-impact layers or execution steps, multiple participants may perform the same computation, allowing mismatches to be detected through lightweight comparison.

* State-local execution: Execution path affinity minimizes state transfer and confines errors to limited execution segments.

These mechanisms allow many incorrect executions to be detected within the same token step or shortly thereafter, enabling rapid fallback or localized re-execution before incorrect results propagate to the user.

10.3. Control Path Functions and Enforcement

The control path operates asynchronously and is responsible for system-wide coordination and accountability. Its functions include, but are not limited to:

* Cryptographic identity registration and authentication,

* Stake locking, management, and release,

* Validation of execution commitments and revealed results,

* Slashing decisions for incorrect execution or missed deadlines,

* Reward calculation and settlement,

* Maintenance of long-term reputation or eligibility signals.

Because control path operations are not latency-critical, they can perform thorough verification and policy enforcement without delaying inference execution. Control path decisions are driven by signed execution records and verifiable evidence produced during inference path execution.

While some violations may be detected only after inference has progressed or completed, economic deterrence and reputational consequences make sustained misbehavior irrational for participants seeking long-term participation.

10.4. Bounded Risk and Practical Correctness

The separation between the inference path and the control path introduces bounded risk rather than absolute prevention of incorrect execution. However, the combination of early detection on the inference path, rapid fallback mechanisms, and strong economic deterrence on the control path ensures that incorrect results are rare, localized, and unlikely to persist undetected.

This design follows a well-established distributed systems principle: optimistic execution combined with eventual verification. ODSI applies this principle to open and decentralized inference, enabling high-throughput, low-latency execution while preserving accountability and system integrity.

11. Scalability and Deployment Considerations

ODSI is designed to scale across large numbers of independently operated participants and to operate under dynamic network and resource conditions. This section discusses considerations related to open participation, participant churn, and long-term extensibility of the framework.

11.1. Open Membership and Sybil Resistance

ODSI supports open membership, allowing any participant to contribute computational resources without requiring centralized admission or prior trust relationships. This openness is essential for elastic scaling and for harnessing widely distributed and heterogeneous compute capacity.

However, open membership introduces the risk that a single entity may create many identities to gain disproportionate influence or rewards. To mitigate this risk, ODSI relies on Sybil-resistance mechanisms implemented within the control plane, where Sybil resistance can only be achieved economically rather than administratively. While identity creation is unrestricted, meaningful participation requires stake-backed commitment. Stake ensures that influence, task volume, and rewards are proportional to economic risk and historical performance rather than identity count.

This approach preserves openness while preventing adversaries from cheaply amplifying influence through large numbers of identities. Large-scale Sybil attacks require correspondingly large capital commitments and carry a high risk of economic loss.

11.2. Growth and Churn

Participants in ODSI may join or leave the system at any time, either intentionally or due to failures, mobility, or network conditions. The framework assumes continuous churn and does not require long-lived availability from individual participants.

Scalability under churn is achieved by:

* Decentralized participant discovery and coordination,

* Adaptive assignment of inference execution based on observed availability and performance,

* Conservative state migration and localized recovery when execution paths change.

The inference plane prioritizes timely progress in the presence of transient failures, while the control plane provides longer-term stability by discouraging unreliable behavior through economic and reputational consequences. Together, these mechanisms allow the system to scale with participation while remaining robust under dynamic conditions.

11.3. Interoperability and Extensibility

ODSI is defined as an architectural framework rather than a single monolithic protocol. It is intended to accommodate multiple protocol instantiations, execution environments, and incentive mechanisms.

Interoperability is supported by:

* Clear separation between inference execution and verification functions,

* Well-defined interfaces between planes,

* Use of cryptographic primitives and message formats that can be standardized independently.

Extensibility is a key design goal. New execution strategies, verification techniques, or incentive schemes can be introduced without disrupting existing deployments, provided they respect the framework宴洉 core principles. This modularity allows ODSI to evolve alongside advances in inference techniques, hardware capabilities, and decentralized coordination mechanisms.

12. Privacy Considerations

Inference execution in ODSI involves the transmission of user-provided inputs and intermediate activations across independently operated participants. By default, the framework does not provide confidentiality guarantees for such data beyond transport-level protection.

Decentralized execution may reduce the need to transmit user data to centralized endpoints and can enable computation to occur closer to data sources. However, distributing execution across multiple participants may also increase the number of entities that observe portions of the execution state.

Privacy risks include exposure of user inputs, partial activations, execution patterns, or metadata such as timing and routing information. These risks vary depending on deployment context, participant selection policies, and execution strategies.

ODSI is designed to be compatible with additional privacy-enhancing mechanisms, including data minimization, execution on trusted hardware, encrypted computation, or differential privacy techniques. Such mechanisms are considered out of scope for this document but may be incorporated by specific protocol instantiations or deployment profiles.

Operators and users should carefully evaluate privacy requirements and select appropriate configurations and extensions when deploying ODSI in sensitive environments.

13. Relationship to Existing Work

ODSI draws inspiration from multiple areas of distributed systems and decentralized computing, while addressing challenges that are specific to large-scale, interactive inference workloads.

Centralized inference platforms provide tightly optimized execution but rely on trusted operators and centralized infrastructure. ODSI departs from this model by enabling open participation and decentralized execution while preserving interactivity.

Prior work on distributed machine learning focuses primarily on training or batch-oriented computation, where execution is less sensitive to strict per-step latency and state continuity. In contrast, ODSI targets online inference with strong sequential dependencies and deadline constraints.

Peer-to-peer and decentralized computation frameworks enable open resource contribution but typically assume best-effort execution or stateless tasks. ODSI extends these ideas by incorporating stateful execution, deadline awareness, and economic accountability.

Blockchain and decentralized ledger systems provide mechanisms for identity, verification, and incentive alignment, but are generally unsuitable for latency-critical execution. ODSI adopts similar accountability principles while decoupling execution from verification to meet performance requirements.

By integrating concepts from these domains, ODSI defines a distinct architectural approach for open, decentralized inference that complements rather than replaces existing systems.

14. Acknowledgments

The authors would like to thank colleagues and reviewers in the community who provided feedback on the early version of this draft.

15. References

15.1. Normative References

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/rfc/rfc6979>.

15.2. Informative References

[Bitcoin] Nakamoto, S., "Bitcoin: A peer-to-peer electronic cash system", 2008.

[Byzantine] Castro, M. and B. Liskov, "Practical byzantine fault tolerance", OSDI, February 1999, <https://www.usenix.org/legacy/publications/library/proceedings/osdi99/full_papers/castro/castro.ps>.

[Lightning] Joseph Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments", 2008.

Authors' Addresses

Hanling Wang
Pengcheng Laboratory
Email: wanghl03@pcl.ac.cn

Qing Li
Pengcheng Laboratory
Email: liq@pcl.ac.cn

Yong Jiang
Tsinghua Shenzhen International Graduate School & Pengcheng Laboratory
Email: jiangy@sz.tsinghua.edu.cn

Mingwei Xu
Tsinghua University
Email: xumw@tsinghua.edu.cn