# EigenAI: Deterministic Inference, Verifiable Results

David Ribeiro Alves
david@eigenlabs.org

Vishnu Patankar
vishnu@eigenlabs.org

Matheus Pereira
matheus@eigenlabs.org

Jamie Stephens
jamie@eigenlabs.org

Nima Vaziri
nima@eigenlabs.org

Sreeram Kannan
sreeram@eigenlabs.org

## Abstract

**EigenAI** is a verifiable AI platform built on top of the EigenLayer restaking ecosystem. At a high level, it combines a deterministic large–language model (LLM) inference engine with a cryptoeconomically secured optimistic re–execution protocol so that every inference result can be publicly audited, reproduced and, if necessary, economically enforced. An untrusted operator runs inference on a fixed GPU architecture, signs and encrypts the request and response, and publishes the encrypted log to EigenDA. During a challenge window, any watcher may request re–execution through EigenVerify; the result is then deterministically recomputed inside a trusted execution environment (TEE) with a threshold–released decryption key, to allow a public challenge with private data. Because inference itself is bit–exact, verification reduces to a byte–equality check and a single honest replica suffices to detect fraud. We show how this architecture yields sovereign agents—prediction–market judges, trading bots, scientific assistants—that enjoy state–of–the–art performance while inheriting security from Ethereum's validator base.

## 1 Introduction and Motivation

Large-language-model (LLM) inference is rapidly evolving from a consumer-facing chatbot interface into a critical back-end service for autonomous and semi-autonomous agents. These agents may trade assets, adjudicate market outcomes, draft contracts, or curate social feeds; in all cases, they must be **trusted**. Today's cloud AI APIs offer impressive performance but provide no cryptographic or economic assurance that an inference was executed faithfully on the claimed model and inputs. This *trust gap* renders current AI infrastructure unsuitable for high-stakes or on-chain contexts.

**Verifiability as a missing primitive.** Blockchains revolutionized finance by making state transitions publicly verifiable and economically final. In contrast, AI systems remain opaque: the mapping from prompt to output is hidden behind proprietary infrastructure, and inference itself is nondeterministic on modern GPUs. Two identical queries to the same model can yield divergent outputs because of floating-point non-associativity, kernel scheduling, and variable batching. Without reproducibility, *verification through re-execution*—the approach underpinning optimistic blockchains—is impossible.

**EigenAI's proposition.** EigenAI closes this gap by introducing a complete verifiable-AI stack:

1. **Deterministic inference:** bit-exact reproducibility on fixed GPU architectures using custom kernels, version-pinned drivers, and canonical reduction orders.
2. **Optimistic verification:** inference results are posted, encrypted, to EigenDA and enter a challenge period. Any verifier can re-execute deterministically; mismatches trigger *slashing* of the operator's stake.
3. **Privacy:** all user prompts and results remain confidential through threshold key management and TEE-based attestation before decryption.
4. **Economic security:** backed by EigenLayer's validator base—millions of restaked ETH—providing orders of magnitude more collateral than bespoke AI networks.

**Sovereign verifiable agents.** On top of this foundation, developers can deploy "sovereign" agents whose logic and reasoning steps are cryptographically traceable. Prediction-market adjudicators, AI traders, scientific analysts, or verifiable NPCs in games can all operate under the same principle: every inference is reproducible, every deviation is detectable, and every misbehavior is penalized.

**Where verifiability matters most.** Verifiable inference is most valuable when an agent's output triggers an *irreversible* external action or resolves a dispute between mutually distrusting parties. Concrete classes of sovereign-agent applications that benefit the most include:

1. **On-chain adjudication and dispute resolution:** prediction markets, insurance claims, and DAO governance that require a publicly auditable ruling rather than a trusted intermediary.
2. **Autonomous execution agents:** trading, liquidation, and treasury-management bots whose actions move real capital and therefore benefit from accountable, replayable decision traces.
3. **Compliance- and audit-driven workflows:** contract drafting, policy enforcement, and scientific/engineering assistants where later auditability ("what was executed, under which model/environment, and why?") is as important as raw model quality.

In these settings, deterministic receipts plus an enforceable challenge process turn an opaque API call into a verifiable, economically accountable computation.

**Paper organization.** Section 1 motivates the need for verifiable inference. Section 2 reviews prior approaches to verifiable computation and deterministic execution. Subsequent sections (not yet populated here) will describe the EigenAI architecture, deterministic-GPU methodology, optimistic-re-execution protocol, economic guarantees, and empirical results.

## 2 Background and Related Work

Three broad paradigms exist for making AI inference verifiable:

**Cryptographic Proofs of Correctness** Zero-knowledge (ZK) and interactive proof systems can, in principle, produce a succinct proof that an untrusted operator executed a neural network faithfully. Systems such as SafetyNets [1] and later zkDNN frameworks [2] demonstrate this feasibility but remain impractical for frontier LLMs: even with hardware acceleration, proving a full transformer forward pass takes minutes to hours. The high cost of circuit synthesis and proof generation limits adoption to small or static models.

**Statistical or Consensus-Based Replication** An alternative is to execute the same query on multiple replicas and accept the majority or the statistically consistent output. Methods include Monte-Carlo dropout and deep ensembles [3, 4] and, more recently, self-consistency decoding [5]. However, these approaches only bound the probability of correctness and cannot detect rare but adversarial divergences [6]. Moreover, their cost scales with $O(\varepsilon^{-2} \log n)$ replicas to achieve error $\varepsilon$—impractical for billion-parameter models.

**Deterministic Execution Environments** Deterministic inference guarantees bit-for-bit identical outputs for identical inputs. CPU or WebAssembly sandboxes (e.g., PyTorch deterministic mode [7], ONNX Runtime Web [8]) provide reproducibility but are 10–100$\times$ slower than GPU back-ends and cannot serve production-scale LLMs. Recent vendor documentation (e.g., NVIDIA cuBLAS reproducibility guide [9, 10]) and research [11, 12] show that determinism on GPUs is attainable if hardware architecture, driver, and library versions are fixed and atomic reductions avoided.

**Optimistic Verification and Crypto-Economic Guarantees** Optimistic rollups in blockchain systems introduced a model where results are accepted by default but can be *challenged* through re-execution; dishonest operators are economically penalized. EigenAI extends this idea to AI inference. Determinism enables disputes to collapse to a simple byte-equality check rather than a full consensus or proof-generation process. EigenVerify—the verification layer—leverages EigenLayer's restaked validator pool to provide the necessary bonded capital for slashing. Because verification is only invoked under dispute, the steady-state cost approaches that of normal inference while maintaining cryptographic accountability.

**Trusted Hardware and Threshold Key Management** Trusted Execution Environments (TEEs) such as Intel SGX or AMD SEV provide hardware isolation and remote attestation [13]. When combined with threshold cryptography, they allow privacy-preserving verification: encrypted requests on EigenDA are decrypted only inside attested enclaves that prove correct code execution. This design mitigates the trade-off between verifiability and confidentiality.

**Summary.** Table 1 summarizes these paradigms by latency, cost, and trust assumptions. EigenAI combines deterministic inference (for fast re-execution) with optimistic crypto-economic enforcement (for security), achieving a unique balance of speed, cost, and trust-minimization.

## 3 System Model and Threats

EigenAI's trust model extends EigenLayer's *Actively Validated Services (AVS)* framework to AI inference. It formalizes how operators, verifiers, and users interact under deterministic execution and cryptoeconomic guarantees. This section defines the system participants, their responsibilities, the security assumptions, and adversarial capabilities.

### 3.1 System Entities

**Client / Requester** Submits an inference request $\mathsf{req}$ consisting of a model identifier, container digest, GPU architecture tag, driver/toolkit version, decoding policy, and prompt commitments. Requests are signed and optionally encrypted to the EigenAI public key before dispatch.

**Operator** Executes inference inside a containerized runtime fixed to a single GPU architecture (e.g., H100). Produces outputs ($\mathsf{out}$, $\mathsf{logits}$), constructs a signed *receipt* committing to input/output hashes, and posts the ciphertext and receipt to EigenDA. Each operator maintains an on-chain identity and bonded stake in EigenLayer.

**EigenDA** A data-availability layer ensuring immutable publication of receipts and ciphertexts. Provides inclusion proofs for challenge adjudication.

**EigenVerify** A decentralized network of verifiers, economically secured by EigenLayer stake, that handles challenges. Each verifier runs a threshold-cryptography Key Management Service (KMS) and trusted execution environment (TEE) runtime. On challenge, it re-executes the inference deterministically to confirm or refute the operator's claim.

**KMS Shards** Hold encrypted key shares for the EigenAI application private key. They release shares only to enclaves that successfully attest correct code identity, enabling privacy-preserving re-execution.

### 3.2 Workflow Overview

At a high level, EigenAI follows an *optimistic* submit–publish–verify pipeline whose correctness hinges on deterministic re-execution. We briefly narrate the end-to-end flow and then detail each phase.

**Submission.** A client constructs and signs an inference request $\mathsf{req}$ that fixes the model, container digest, GPU architecture, driver/toolkit version, decoding policy, PRNG seed, and (optionally) prompt commitments. The signed $\mathsf{req}$ is transmitted to an operator for execution. In practice, treating these fields as *immutable execution parameters* is what later allows any verifier to replay the request under identical conditions.

**Execution.** Upon receipt, the operator runs the model under the declared environment, producing the output $\mathsf{out}$ together with (optionally) auxiliary artifacts such as per-step logits. Because the execution stack is deterministic (cf. Section 6), any honest rerun of the same request on the same architecture will yield a byte-identical $\mathsf{out}$.

**Publication.** To preserve confidentiality while enabling public audit, the operator encrypts $(\mathsf{req}, \mathsf{out})$ to the application public key $\mathsf{pk}_{\mathsf{app}}$ and posts the resulting ciphertext, together with a signed *receipt* $\sigma_{\mathsf{op}}$, to EigenDA. The receipt canonically commits to the request and output via their hashes and may include a TEE attestation quote and timestamp, along with a durable pointer to the DA record. This publication anchors both *availability* (via EigenDA) and *integrity* (via the operator's signature and the receipt fields) of the claimed execution.

**Challenge window.** Published results are tentative for a fixed dispute horizon of $\Delta$ epochs. During this window, any party may inspect receipts and either initiate a low-cost *light audit* or file a formal *full challenge*. The former offers probabilistic coverage without slashing authority; the latter invokes on-chain adjudication and possible penalties.

**Re-execution and voting.** When a full challenge is raised, EigenVerify samples a stake-weighted committee of verifiers. Each verifier boots an attested TEE running the approved container, establishes mutually attested channels to KMS shards to reconstruct $\mathsf{sk}_{\mathsf{app}}$ *inside* the enclave, decrypts the EigenDA ciphertext, and deterministically re-executes $\mathsf{Infer}(\mathsf{req})$. The committee then decides by *byte-equality vote*: each member casts $b_v = [\widehat{\mathsf{out}}_v = \mathsf{out}]$, and the verdict is determined by threshold (e.g., $\geq 2/3$).

**Finalization.** If the committee agrees that $\widehat{\mathsf{out}} = \mathsf{out}$, the result is finalized; otherwise, the operator is slashed and the committee's majority output replaces the disputed one. This optimistic design amortizes

Table 1: Comparison of verifiable-inference paradigms.

| **Paradigm** | **Latency** | **Cost** | **Guarantee** |
|---|---|---|---|
| ZK proofs | very high | high | mathematical |
| Statistical replication | medium | high (many replicas) | crypto-economical |
| CPU/WASM Det. | very high | low | crypto-economical |
| GPU determinism + optimism (EigenAI) | low | low | crypto-economical |

cost—verification runs only under dispute—while determinism collapses adjudication to a binary equality check.

## 3.3 Security Assumptions

The security of the protocol rests on standard, explicit assumptions that align with its layered design:

- **Deterministic execution.** Holding fixed the GPU architecture, driver, toolkit, and decoding policy, repeated runs of $\mathsf{Infer}(\mathsf{req})$ are bit-identical (Section 6). This guarantees that honest re-executions converge to a unique $\widehat{\mathsf{out}}$.
- **Data availability.** EigenDA provides durable storage and inclusion proofs for all posted receipts and ciphertexts, ensuring that disputes can always retrieve the exact bytes committed at publication.
- **Stake honesty.** During any challenge epoch, at least two thirds of EigenVerify stake behaves honestly. This Byzantine-style assumption underwrites the committee vote and the credibility of slashing events.
- **TEE integrity.** Verifier enclaves support remote attestation that binds code identity (container digest) and GPU mode to a measurement; only enclaves presenting valid quotes may participate in decryption and re-execution.
- **Threshold confidentiality.** The EigenAI application private key is $t$-of-$n$ secret-shared across KMS shards; fewer than $t$ colluding shards learn nothing useful, and shares are released only to enclaves that satisfy attestation policy.

## 3.4 Adversary Model

We consider a powerful adversary that may compromise some off-chain components, subject to the assumptions above:

1. *Dishonest operator.* Attempts to report falsified outputs, substitute models/containers, or replay stale receipts in lieu of fresh computation.
2. *Colluding verifiers.* A minority of EigenVerify stake coordinates to bias votes, delay challenges, or attempt to exfiltrate plaintext via misconfigured enclaves.
3. *Compromised KMS shard.* A single (or minority) shard discloses partial key material or responds to non-attested endpoints.
4. *Malicious DA participant.* Censors or withholds ciphertexts/receipts to prevent effective challenges or inclusion proof verification.
5. *Timing/side-channel attacker.* Observes or perturbs enclave execution to infer private data or influence control flow without altering code identity.

## 3.5 Threats and Mitigations

Table 2 consolidates the principal threat classes with their first-line defenses. In combination—deterministic kernels and pinned environments (technical reproducibility), on-chain receipts and DA proofs (cryptographic integrity), TEEs and threshold KMS (confidentiality), and stake-backed slashing (economic deterrence)—the system achieves layered, defense-in-depth protection.

# 4 Protocol Overview

EigenAI implements an *optimistic*, verifiable inference pipeline in which results are accepted by default but can be efficiently disputed and re-executed under cryptoeconomic guarantees. In what follows, we present the submission path, the receipt and data-availability interface, the audit and challenge flows, and the deterministic re-execution procedure that together realize this trust model.

## 4.1 Submission and Dataflow

Each inference traverses a structured lifecycle (Fig. 1). The design principle is that *every parameter that can influence numerical outcomes is fixed and committed*

Table 2: Primary threat classes and mitigations.

| Threat | Mitigation |
|---|---|
| Model / kernel tampering | Immutable container digests; signed receipts; open-source deterministic kernels. |
| Cross-architecture drift | Single-arch policy per request; explicit `gpu_arch` field in receipt; verifier enforcement. |
| Library / driver drift | Container pinned by digest; toolkit and driver version fixed; GPU clock locking. |
| Batch nondeterminism | Batch-invariant reduction kernels; fixed decode seeds. |
| KMS compromise | $t$-of-$n$ threshold policy; attestation-gated share release. |
| TEE compromise | Hardware attestation; code measurement checks; enclave-to-shard TLS. |
| Verifier collusion | Stake-weighted majority voting; light audits; fork-choice backstop. |
| Data withholding (DA) | EigenDA inclusion proofs; redundancy across operators. |

*up front*, enabling any verifier to replay the request in an identical environment.

**1. Request preparation.** The client constructs a canonical request

$$\mathsf{req} = \left\langle \begin{array}{c}\texttt{container\_digest}, \texttt{gpu\_arch}, \texttt{driver\_tag},\\ \texttt{decode\_policy}, \texttt{seed}, \texttt{prompt\_commitments}\end{array} \right\rangle.$$

signs it, and submits it to an operator. All fields are treated as immutable execution parameters for reproducibility; in particular, `prompt_commitments` (when present) is a Merkle root that binds any external documents or tool outputs referenced by the prompt.

**2. Deterministic execution.** The operator runs the model inside the declared container on the declared hardware architecture, producing the token sequence and, optionally, per-step logits. By construction (Section 6), this execution is *bit-deterministic*: rerunning the same request under the same environment yields the identical byte stream.

**3. Receipt formation and publication.** To couple confidentiality with auditability, the operator encrypts $(\mathsf{req}, \mathsf{out})$ to the application public key $\mathsf{pk}_{\mathsf{app}}$ and posts the ciphertext together with a signed receipt to EigenDA. The receipt commits to the request and output via their hashes and may include attestation evidence and timing metadata:

$$\mathsf{receipt} = \left\langle \begin{array}{c} H(\mathsf{req}), H(\mathsf{out}), \texttt{model\_id},\\ \texttt{chainid}, \texttt{da\_pointer}\end{array} \right\rangle, \sigma_{\mathsf{op}} = \mathrm{Sign}_{sk_{\mathsf{op}}}(\mathsf{receipt}).$$

Publishing to EigenDA establishes durable availability (for future disputes), while the operator's signature anchors integrity and provenance.

**4. Data availability and challenge window.** Upon publication, the result enters a fixed dispute horizon of $\Delta$ blocks. During this *challenge window*, any party may retrieve the receipt and either perform a low-cost *light audit* or lodge a formal *full challenge*. The former offers randomized coverage without on-chain penalties; the latter triggers adjudication with the possibility of slashing.

## 4.2 Light Audit versus Full Challenge

**Light audit.** A user or watchdog recruits a small, randomly chosen subset of EigenVerify nodes to re-execute the request off-chain. This provides probabilistic assurances at minimal cost and is well-suited for continuous, background integrity monitoring.

**Full challenge.** If an inconsistency is detected—or if a counterparty disputes a result—an on-chain challenge is filed. EigenVerify then samples a stake-weighted committee $\mathcal{V}$ representing a supermajority of bonded capital. Committee members re-execute the request in attested TEEs and vote by byte equality on the operator's claim.

## 4.3 Deterministic Re-Execution and Voting

The adjudication step consists of *reproducing* the claimed computation inside trusted enclaves and deciding by equality of bytes. Concretely, each verifier $v \in \mathcal{V}$:

1. boots an enclave with the approved container (producing an attestation quote),
2. proves attestation to the KMS shards and reconstructs $\mathsf{sk}_{\mathsf{app}}$ *in-enclave*,
3. fetches the ciphertext and receipt from EigenDA and verifies $\sigma_{\mathsf{op}}$,
4. deterministically runs $\mathsf{Infer}(\mathsf{req})$ to obtain $\widehat{\mathsf{out}}_v$,
5. casts a vote $b_v = [\,\widehat{\mathsf{out}}_v = \mathsf{out}\,]$.

If the vote fraction satisfies $\sum b_v/|\mathcal{V}| \geq \tau$ (e.g., $\tau = 2/3$), the result is accepted; otherwise, the operator is slashed and the committee's majority output replaces the disputed one. Determinism collapses the decision to a binary equality test, eliminating ambiguity and extensive deliberation.

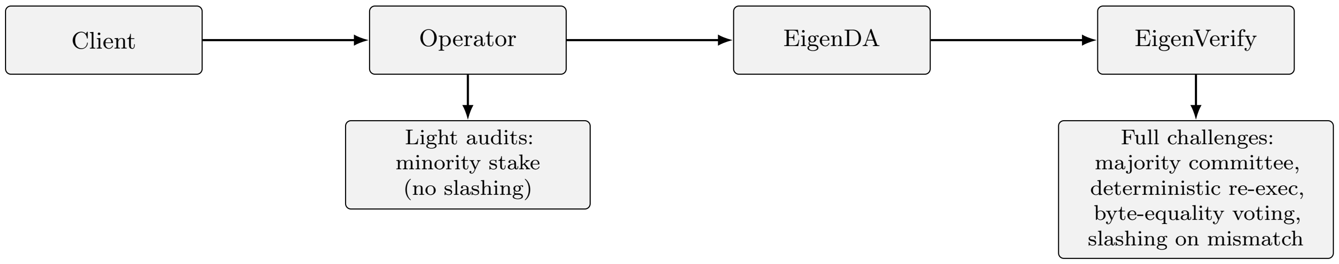
Client
Operator
EigenDA
EigenVerify
Light audits: minority stake (no slashing)
Full challenges: majority committee, deterministic re-exec, byte-equality voting, slashing on mismatch

Figure 1: Swimlane depicting Client → Operator → EigenDA → EigenVerify. Light audits sample a small minority of stake (no slashing); full challenges invoke a majority committee for deterministic re-execution, byte-equality voting, and slashing on mismatch.

Table 3: Receipt schema and field semantics.

| Field | Purpose |
|---|---|
| `model_id` | Identifier of model weights |
| `chain_id` | Identifier of the chain |
| `gpu_arch` | Hardware generation (e.g., H100) |
| `req_hash` | Hash of request parameters |
| `out_hash` | Hash of output logits/tokens |
| `da_pointer` | EigenDA inclusion reference |
| `sig` | Operator's digital signature |

**Algorithm 1** Operator submission routine (canonicalized).

```
1: Input: $\mathsf{req}$, model $\mathsf{M}$, container $\mathsf{C}$, GPU arch $a$
2: $\mathsf{out} \leftarrow \mathsf{Infer}(\mathsf{M}, \mathsf{C}, a, \mathsf{req})$
3: $\mathsf{rcp} \leftarrow \langle H(\mathsf{req}),\ H(\mathsf{out}),\ t\rangle$
4: $\sigma_{\mathsf{op}} \leftarrow \mathsf{Sign}_{sk_{\mathsf{op}}}(\mathsf{rcp})$
5: $\mathsf{cipher} \leftarrow \mathsf{Enc}_{pk_{\mathrm{app}}}(\mathsf{req}, \mathsf{out})$
6: Publish $(\mathsf{cipher}, \mathsf{rcp}, \sigma_{\mathsf{op}})$ to EigenDA
7: Start challenge timer $\Delta$
```

*Cost amortization.* Because re-execution is invoked only under dispute, steady-state operation mirrors ordinary inference costs. When challenges do occur, determinism ensures that even a single honest verifier suffices to detect fraud, and a small committee can finalize outcomes with minimal overhead.

## 5 Privacy Architecture: Threshold KMS and TEEs

While verifiability necessarily promotes transparency, many EigenAI users operate on sensitive data that must remain private. To reconcile these opposing demands, EigenAI layers a robust *confidentiality substrate* atop its verifiable infrastructure through a combination of threshold key management and trusted execution environments (TEEs). This architecture allows verification of correctness without revealing the underlying user data.

**Algorithm 2** Full challenge verification (deterministic re-execution).

```
1: Input: DA pointer $p$, receipt $\mathsf{rcp}$, signature $\sigma_{\mathsf{op}}$
2: $\mathcal{V} \leftarrow$ stake-weighted committee sample
3: for $v \in \mathcal{V}$ in parallel do
4:     Boot attested enclave; obtain quote $q_v$
5:     Establish attested channels to KMS; reconstruct $\mathsf{sk}_{\mathsf{app}}$
6:     Download $(\mathsf{cipher}, \mathsf{rcp})$ from EigenDA
7:     Verify $\sigma_{\mathsf{op}}$; decrypt to $(\mathsf{req}, \mathsf{out})$
8:     $\widehat{\mathsf{out}}_v \leftarrow \mathsf{Infer}(\mathsf{req})$
9:     Vote $b_v \leftarrow [\,\widehat{\mathsf{out}}_v = \mathsf{out}\,]$
10: end for
11: if $\sum b_v/|\mathcal{V}| < \tau$ then
12:     Slash operator stake; finalize majority $\widehat{\mathsf{out}}$
13: else
14:     Finalize $\mathsf{out}$ as verified
15: end if
```

### 5.1 End-to-End Encryption and Key Management

Every inference request and its corresponding output are encrypted to the EigenAI application public key $\mathsf{pk}_{\mathsf{app}}$ before publication. The corresponding private key $\mathsf{sk}_{\mathsf{app}}$ is never held in a single location; instead, it is fragmented into $n$ shares and distributed across the EigenVerify Key Management Service (KMS) network using a $t$-of-$n$ threshold scheme, such as Shamir's secret sharing. No single KMS shard can decrypt or reconstruct $\mathsf{sk}_{\mathsf{app}}$ independently, and shards only release key shares to enclaves that successfully prove their authenticity and code integrity via remote attestation. This design enforces that decryption can occur *only within verified, attested enclaves*, ensuring that plaintext data never exists outside of secure execution contexts.

### 5.2 Remote Attestation and Secure Share Release

The interaction between verifier enclaves and KMS shards follows a mutually authenticated sequence, depicted conceptually in Fig. 2. This sequence guarantees that key material is distributed only to legitimate enclaves running approved EigenAI software stacks:

1. A verifier launches a TEE running the approved container image, producing a hardware-signed attestation quote $q$ that includes a cryptographic hash of the loaded binary (the *measurement*).
2. Each KMS shard validates $q$, confirming that the enclave is both genuine and running an authorized EigenAI image. Quotes are also checked for freshness to prevent replay attacks.
3. After successful validation, shards establish mutually attested TLS sessions with the enclave, ensuring end-to-end confidentiality and integrity of communication.
4. Shards transmit their encrypted key shares to the enclave, which reconstructs $\mathsf{sk}_{\mathsf{app}}$ entirely in volatile memory. Using this key, the enclave decrypts the EigenDA ciphertext and proceeds with deterministic re-execution of the inference task.
5. Upon completion, the enclave securely zeroizes $\mathsf{sk}_{\mathsf{app}}$ and all session-specific secrets, preventing residual key material from persisting after verification.

This remote attestation sequence is central to EigenAI's privacy architecture: it cryptographically binds data access to verified code identity, thereby eliminating the possibility of decryption by compromised nodes or untrusted software.

### 5.3 Auditability Without Decryption

Importantly, confidentiality does not come at the expense of transparency. Because each inference is accompanied by cryptographic commitments—$H(\mathsf{req})$ and $H(\mathsf{out})$—external auditors can verify inclusion proofs on EigenDA and validate operator signatures without accessing any plaintext data. This property allows independent parties to conduct statistical audits of operator honesty and data-availability compliance while maintaining end-to-end encryption of user content. In effect, the system preserves both *verifiable correctness* and *privacy by design*.

### 5.4 Key Epochs, Rotation, and Policy Enforcement

To further mitigate long-term compromise risk, EigenAI enforces periodic key rotation through *key epochs*. Each receipt explicitly records the key epoch used during encryption. KMS policies track these epochs and automatically deny reconstruction requests for retired keys. When a rotation event occurs—either on schedule or triggered by a security incident—new key shares are generated, and governance proposals via EigenVerify are used to update metadata across participants. This guarantees forward secrecy while maintaining uninterrupted availability for active requests.

Taken together, these mechanisms ensure that verifiability and confidentiality coexist harmoniously. Deterministic execution and public receipts make correctness independently checkable, while threshold cryptography and attested enclaves guarantee that user data remains inaccessible to all parties except during secured, ephemeral re-execution inside TEEs.

## 6 Deterministic Inference: Technical Foundations

Deterministic inference forms the *technical cornerstone* of EigenAI's verifiability framework. Without strict bit-level reproducibility, optimistic re-execution would become ambiguous—disputes could not be resolved by simple equality checks, and consensus would devolve into probabilistic agreement. This section surveys the sources of nondeterminism in modern GPU-based deep-learning systems, outlines the engineering controls used to eliminate them, and discusses their empirical validation. It extends our prior *Bit-Exact Inference on GPUs* work with new insights specific to cryptoeconomic verification.

### 6.1 Why Determinism Matters

Large language model (LLM) inference comprises thousands of parallel GPU kernels performing linear algebra and nonlinear reductions. Minute variations in operation ordering, rounding behavior, or kernel selection can perturb the resulting logits and, consequently, alter sampled tokens. In everyday applications this variability is imperceptible; in a verifiable execution setting it is catastrophic. Because EigenVerify relies on comparing the outputs of independent re-executions, even a single bit of nondeterministic drift would undermine the ability to distinguish honest disagreement from dishonesty.

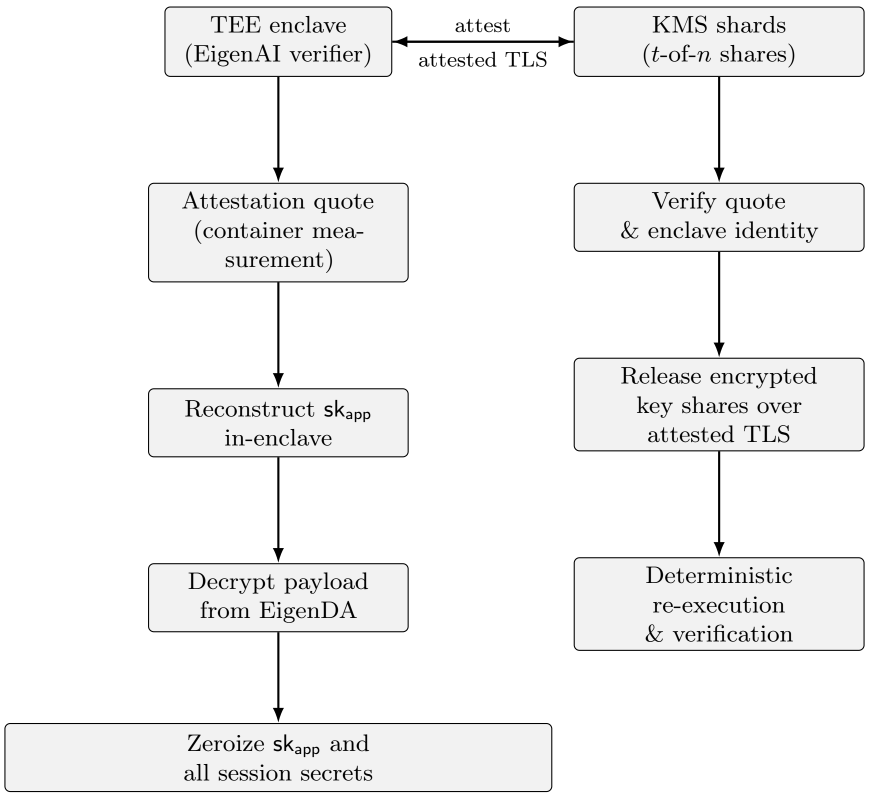
TEE enclave
(EigenAI verifier)
attest
attested TLS
KMS shards
($t$-of-$n$ shares)
Attestation quote (container measurement)
Verify quote & enclave identity
Reconstruct $\mathsf{sk}_{\mathsf{app}}$ in-enclave
Release encrypted key shares over attested TLS
Decrypt payload from EigenDA
Deterministic re-execution & verification
Zeroize $\mathsf{sk}_{\mathsf{app}}$ and all session secrets

Figure 2: TEE–KMS negotiation flow. The enclave attests its container measurement; KMS shards verify the quote, establish attested TLS connections, and release key shares. The enclave reconstructs $\mathsf{sk}_{\mathsf{app}}$ in memory, decrypts the payload, performs verification, and zeroizes all secrets afterward.

Establishing determinism transforms inference from a stochastic numerical process into a pure function:

$$\mathcal{F} : (\mathsf{model}, \mathsf{arch}, \mathsf{prompt}, \mathsf{seed}, \mathsf{decode}) \longrightarrow \mathsf{output},$$

where $\mathsf{output}$ is guaranteed to be bit-identical for all honest re-executions given the same parameters.

## 6.2 Sources of Nondeterminism

Determinism in GPU inference is fragile and may be compromised by variations across the hardware and software stack. We categorize four principal layers of variability, illustrated conceptually in Fig. 3.

1. **Hardware.** Floating-point units differ slightly across GPU generations (e.g., A100 vs. H100), implementing fused multiply-add (FMA) and rounding modes with subtle architectural distinctions. Deterministic inference therefore requires enforcing single-architecture policies.
2. **Math Libraries.** Core libraries such as cuBLAS, cuDNN, or TensorRT may invoke atomic operations or rely on non-associative accumulation orders, both of which compromise reproducibility. Furthermore, "fast math" and mixed-precision modes often trade consistency for throughput.
3. **Inference Engine.** Framework-level optimizations—dynamic graph fusion, asynchronous kernel launches, and stochastic decoding—introduce another layer of variability. Although frameworks such as PyTorch and TensorFlow offer deterministic flags, these apply only to a subset of supported operations.

## 6.3 Hardware-Level Determinism

Modern NVIDIA GPUs can achieve bit-exact reproducibility when operated under controlled conditions. The Hopper architecture family (H100, GH200) guarantees repeatable outputs from cuBLAS routines on identical GPUs and toolkit versions [10, 9]. Independent investigations confirm that discrepancies between architectures stem primarily from software scheduling, not from arithmetic pipelines [12, 11].

EigenAI enforces a *single-architecture policy* within each deployment: all operators and verifiers must

Table 4: Visibility matrix for EigenAI's privacy and verification components.

| Component | Access to Plaintext? | Description |
|---|---|---|
| Client | Yes | Originator and owner of prompts and outputs. |
| Operator | No | Encrypts all data to $\mathsf{pk}_{\mathsf{app}}$; never sees plaintext. |
| EigenDA | No | Stores ciphertext and receipts; provides inclusion proofs only. |
| KMS Shard | No | Holds encrypted key shares; cannot reconstruct full key. |
| TEE Verifier | Yes (in-enclave) | Attested enclave temporarily reconstructs $\mathsf{sk}_{\mathsf{app}}$ for verification |
| External Auditor | No | Validates hashes, receipts, and signatures without accessing plaintext. |

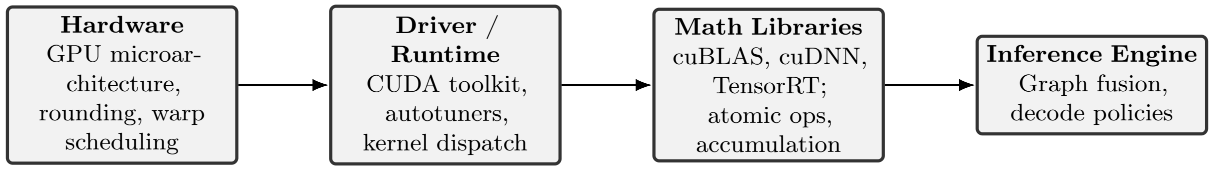
Hardware
GPU microarchitecture, rounding, warp scheduling
Driver / Runtime
CUDA toolkit, autotuners, kernel dispatch
Math Libraries
cuBLAS, cuDNN, TensorRT; atomic ops, accumulation
Inference Engine
Graph fusion, decode policies

Figure 3: Sources of nondeterminism across the GPU stack: (1) Hardware microarchitecture, (2) Driver/runtime, (3) Math libraries and kernels, (4) Inference engine and decode policy. Each layer must be pinned or replaced with deterministic equivalents.

utilize identical GPU SKUs, while persistence mode is enabled to avoid state transitions that might alter kernel execution order.

### 6.4 Determinism in Base Libraries

The inference engine underpinning EigenAI builds upon `llama.cpp`, an open-source C/CUDA implementation with a small and auditable numerical surface. Its quantized matrix-multiplication kernels (e.g., Q4, Q5) are inherently deterministic: they avoid atomics and implement warp-synchronous reductions with fixed thread order. For remaining operations that delegate to cuBLAS or cuBLASLt, EigenAI enforces deterministic configuration flags [25]:

These settings forbid nondeterministic atomics and non-associative mixed-precision reductions. Although cuBLAS is proprietary, its deterministic guarantees have been repeatedly validated in empirical testing.

### 6.5 Deterministic Kernel Design

At the core of EigenAI's reproducibility efforts are custom GEMM kernels and reduction primitives that enforce deterministic ordering. Each kernel satisfies three invariants:

**1. Fixed block–thread mapping.** Thread blocks are deterministically mapped to output tiles with no inter-block communication, ensuring that the GPU's scheduler cannot affect numerical outcomes.

**2. Warp-synchronous reductions.** Within each block, threads compute partial dot-products and perform a canonical binary-tree reduction using warp intrinsics:

```
for (int off = warpSize/2; off > 0; off /= 2)
  sum += __shfl_down_sync(0xffff, sum, off);
```

The reduction order is identical across runs, guaranteeing reproducible rounding paths [26, 27].

**3. No floating-point atomics.** All accumulations are explicitly ordered through register or shared-memory operations. Floating-point atomics are entirely disabled because their non-associative semantics can yield nondeterministic results. Despite this restriction, deterministic kernels maintain 95–98% of standard cuBLAS throughput on Hopper-class hardware.

### 6.6 Deterministic Decoding and PRNG Control

Token generation, which often involves sampling from probability distributions (e.g., top-$k$ or nucleus sampling), introduces another source of variability. EigenAI enforces deterministic decoding by employing a fixed-seed pseudorandom number generator (PRNG) and canonical iteration order. For any pair (`seed`, `decode_policy`), the emitted token sequence is reproducible. Users seeking nondeterministic sampling may simply vary the seed but can still verify that any output matches the declared seed and policy recorded in the operator's receipt.

### 6.7 End-to-End Determinism Experiments

We validated EigenAI's deterministic guarantees through systematic experiments on NVIDIA Hopper GPUs. Each test identical container digests, and consistent runtime environments.

**Setup.** Two independent H100 nodes, both executing `llama.cpp`-based inference, processed a 1,000-prompt benchmark spanning summarization, reasoning, and code generation tasks. For each execution we recorded the hash:

$$\text{SHA256}(\mathsf{prompt} \,||\, \mathsf{logits} \,||\, \mathsf{tokens}).$$

**Results.** Across 10,000 runs, all hashes matched exactly—no bit-level divergence was observed. Cross-architecture comparisons (A100 vs. H100) yielded measurable but expected deviations ($\sim 10^{-7}$ in logits), confirming architecture-dependent rounding and motivating per-architecture verifier pools.

### 6.8 Reproducibility and Verifiability

Determinism collapses verification to a simple equality check. Because every honest re-execution yields an identical byte string, the verification predicate

$$\text{Verify}(\mathsf{out}_1, \mathsf{out}_2) = \begin{cases} \texttt{True}, & \mathsf{out}_1 = \mathsf{out}_2, \\ \texttt{False}, & \text{otherwise} \end{cases}$$

becomes both sound and complete. This property enables EigenAI to scale: verification of thousands of inference tasks reduces to constant-time byte comparisons rather than probabilistic voting or cryptographic proofs.

### 6.9 Implementation Guidelines

For practitioners deploying deterministic inference under EigenAI, the following guidelines are mandatory:

- Pin exact CUDA and driver versions (e.g., CUDA 12.4 with R550 driver).
- Reference container images by digest and avoid mutable tags.
- Enable persistence mode.
- Enable deterministic modes in cuBLAS/cuBLASLt and disallow atomics.
- Disable all nondeterministic math primitives and autotuners.
- Seed PRNGs deterministically and record seeds in receipts.
- Hash and sign $(\mathsf{prompt}, \mathsf{out})$ tuples for auditability

### 6.10 Discussion

Our experiments confirm that bit-exact determinism is achievable on contemporary GPU hardware with negligible performance loss. By constraining variability at every level of the software and hardware stack, EigenAI converts opaque numerical computation into a reproducible process amenable to independent re-execution. This engineering discipline is what enables cryptoeconomic assurance: in EigenAI, correctness can be proven by anyone through mere repetition, with no reliance on statistical tests or zero-knowledge proofs.

## 7 Implementation Details and Developer Experience

EigenAI is designed to feel like a familiar cloud AI service while embedding deterministic and verifiable execution at every layer. Developers interact through an OpenAI-compatible API, and each response carries deterministic metadata, a cryptographic receipt, and a pointer into EigenDA for later verification.

### 7.1 API Compatibility and Metadata

The EigenAI API mirrors the `/v1/chat/completions` and `/v1/completions` endpoints used by OpenAI. Clients can substitute the EigenAI base URL without changing request syntax. Responses contain deterministic metadata fields as shown in Table 6.

This metadata allows any verifier to retrieve the corresponding entry from EigenDA and re-run the request under the same environment.

### 7.2 Container and Hardware Constraints

All inference containers are built atop fixed CUDA/driver pairs, referenced by digest. Operators must enable persistence mode and turn on ECC memory (During the testing phase we discovered undeterminism due to faulty memory. This problem was mitigated by making sure that ECC was turned on). Container and driver versions are registered on-chain and verified by EigenVerify committees during challenges.

Table 5: Empirical determinism verification on Hopper GPUs.

| **Test Condition** | **Match Rate** | **Notes** |
|---|---|---|
| Same host, same GPU | 100.0% | Bitwise identical outputs |
| Same host, same GPU arch but diff number of GPUs | 100.0% | Bitwise identical outputs |
| Different hosts, same cpu arch/GPU SKU | 100.0% | Bitwise identical outputs |
| Different GPU architecture (A100 vs. H100) | 0.0% | Diff. exec paths, rounding |

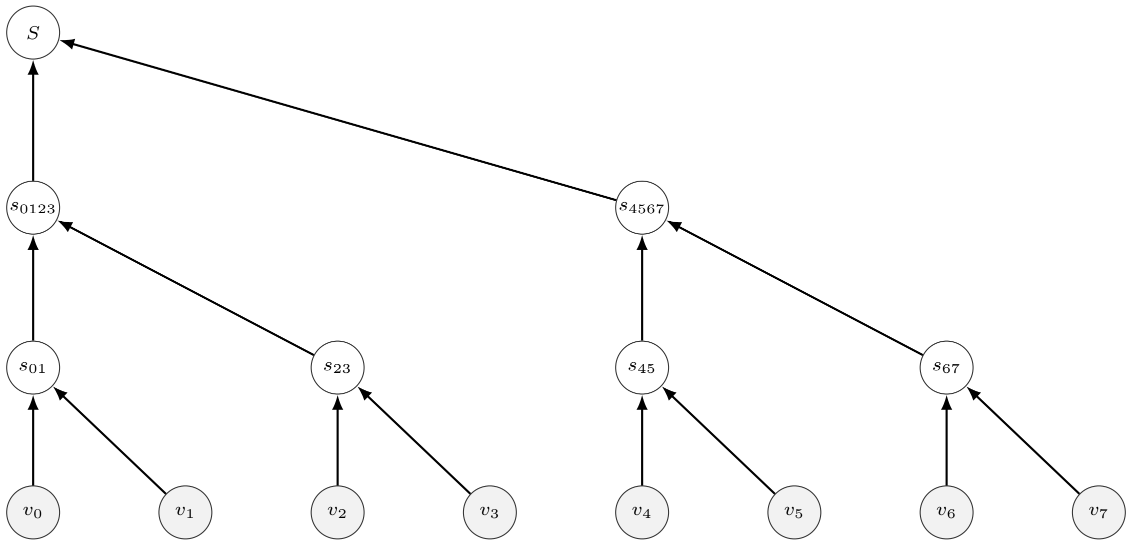

Figure 4: Canonical warp-level reduction tree used in deterministic kernels. Each thread contributes a partial value $v_i$ and participates in pairwise summations in a fixed binary-tree pattern, ensuring identical accumulation order and reproducible results across executions.

**Algorithm 3** Reproduce-and-Verify Procedure

```
1: Input: EigenDA pointer $p$, metadata $\mathsf{meta}$ from API response
2: Download $(\mathsf{cipher}, \mathsf{receipt})$ from EigenDA
3: Verify operator signature $\sigma_{\mathsf{op}}$
4: Launch container $\mathsf{C}$ with exact digest and driver
5: Set environment variables per $\mathsf{meta.system\_fingerprint}$
6: Run $\widehat{\mathsf{out}} \leftarrow \mathsf{Infer}(\mathsf{req}, \mathsf{seed}, \mathsf{decode})$
7: Compare SHA256($\widehat{\mathsf{out}}$) with $\mathsf{receipt.out\_hash}$
8: if hashes match then return VERIFIED
9: elsereturn INVALID
10: end if
```

## 7.3 Reproduction Cookbook for Auditors

Auditors can independently reproduce any inference using the following minimal procedure (Algorithm 3). Because inference is deterministic, matching hashes suffice to validate correctness.

Auditors may also verify EigenDA inclusion proofs to ensure the operator's record was properly published and unaltered.

## 8 Economic and Governance Mechanics

EigenAI inherits its security not only from deterministic execution but also from the broader cryptoeconomic foundation provided by EigenLayer. The economic layer determines how honest behavior is incentivized, how disputes are resolved, and how protocol parameters evolve. In this section we outline the lightweight audit pathway, the full challenge-and-slashing mechanism, and the governance structures that maintain long-term system health.

### 8.1 Light Audits

Light audits provide an inexpensive integrity check on the behavior of operators. A watcher or client may recruit a small, randomly selected subset of EigenVerify nodes to re-execute a published inference off-chain. Be-

Table 6: Key response metadata for deterministic verification.

| Field | Description |
|---|---|
| `system_fingerprint` | Concatenation of container digest, GPU arch, driver version |
| `determinism.seed` | Fixed PRNG seed used for decoding |
| `receipt.req_hash` | SHA256 of request parameters |
| `receipt.out_hash` | SHA256 of model output |
| `receipt.sig` | Operator signature over receipt tuple |
| `eigendalink` | Pointer to EigenDA inclusion proof |

cause these audits lack slashing authority, they impose minimal cost and latency overhead. Their purpose is statistical: by maintaining a nonzero background probability of inspection, they deter latent collusion and encourage operators to remain honest even when they believe they are not under direct scrutiny. Light audits may be rewarded through small bounties or micro-incentives funded by EigenAI usage fees.

## 8.2 Full Challenges and Slashing

A full challenge is invoked when a receipt is formally disputed. EigenVerify samples a stake-weighted committee $\mathcal{V}$ and requires a supermajority (e.g., $> 2/3$) agreement to finalize the result. Each verifier re-executes the inference inside an attested enclave and votes on byte-level equality with the operator's output. A mismatch triggers slashing of the operator's bonded stake, which is redistributed to challengers and verifiers:

$$\text{Reward}_{\text{challenger}} = \alpha S_{\text{slash}}, \qquad \text{Reward}_{\text{verifier}} = \beta S_{\text{slash}},$$

with $\alpha$ and $\beta$ set by governance. Remaining stake may be burned or returned to the EigenLayer treasury.

Because the cost of verification is low compared to potential fraud gains, the expected utility of cheating becomes negative for any reasonable challenge probability $\pi_c$:

$$\mathbb{E}[\text{Gain}] = (1 - \pi_c)G - \pi_c S_{\text{slash}} < 0,$$

where $G$ denotes the maximum benefit from dishonesty. Since $\pi_c$ is augmented by both light audits and user-initiated challenges, rational operators are economically driven to behave honestly.

## 8.3 Fork-Choice Backstop

If an extreme collusion scenario were ever to push an invalid result through verification, EigenLayer's fork-choice rule provides a final safety net. Restakers may coordinate a social fork to penalize misbehaving validators, restoring correctness. This mechanism ensures *economic finality of truth*: the equilibrium strategy for long-term actors is always to preserve correctness rather than collude.

## 8.4 Governance and Parameterization

EigenAI's operational parameters—stake requirements, slashing fractions, challenge thresholds, and audit frequencies—are governed through the EigenLayer governance process. Governance proposals may tune these values over time as workloads, economic conditions, or adversarial models evolve. Looking ahead, governance may also introduce dynamic fee markets for audit capacity, enabling users to purchase higher integrity assurance on demand.

## 9 Security Analysis

We now examine EigenAI's security properties in aggregate, showing how determinism, confidentiality, data availability, and economic incentives interact to form a cohesive and robust trust model.

## 9.1 Security Properties and Enabling Features

We separate *security properties* (what the system should guarantee) from *features of the construction* (engineering choices that help realize those guarantees).

**Desired security properties.** EigenAI targets the following security properties for each published inference:

- **Integrity (correctness):** the published output `out` corresponds to the unique result of running the declared model and request under the committed execution parameters.
- **Confidentiality (privacy):** prompts and outputs remain hidden from unauthorized parties; plaintext

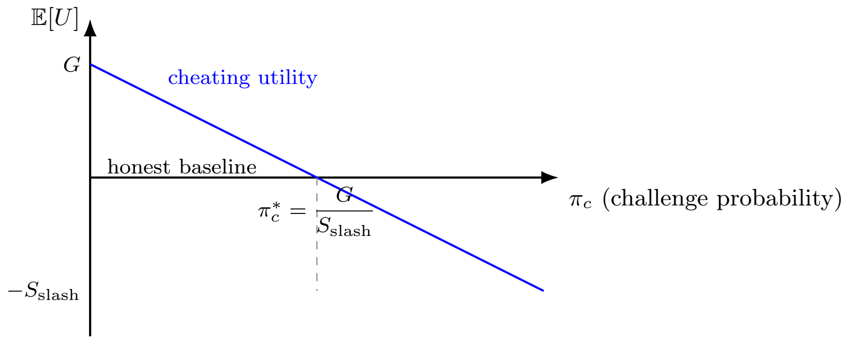

Figure 5: Payoff diagram comparing operator utilities under varying challenge probabilities $\pi_c$. A dishonest operator's expected utility becomes negative once $\pi_c > G/S_{\text{slash}}$, making cheating economically irrational.

is exposed only to authorized clients and, during dispute resolution, transiently inside attested verifier enclaves.

- **Availability:** the evidence needed to audit or dispute an inference (ciphertext, receipt, and DA inclusion evidence) remains retrievable throughout the challenge window.
- **Accountability:** if an operator publishes an incorrect result, there exists a publicly checkable procedure that can penalize the operator (slashing) and finalize a correct outcome.

**Enabling features of the construction.** These properties are supported by (non-exhaustively):

- **Compute determinism** (Section 6), which makes inference effectively single-valued and enables unambiguous re-execution.
- **Cryptographic commitments and data availability** (operator receipts and EigenDA publication), which bind claims to immutable bytes retrievable for disputes.
- **TEE-based private re-execution with threshold keys** (Section 5), which permits verification on private data without revealing plaintext in steady state.
- **Optimistic verification with stake-backed slashing**, which turns detected mismatches into enforceable economic penalties.

## 9.2 Soundness and Completeness

**Soundness.** Soundness requires that dishonest behavior be detectable. If an operator deviates from the canonical deterministic execution, any honest verifier will compute a different output during re-execution. Because verification reduces to byte-equality, disagreement is unambiguous, and the probability of undetected fraud falls exponentially with the fraction of honest stake participating in the committee.

**Completeness.** Completeness requires that honest operators never be penalized. Determinism guarantees that re-executions match the operator's output exactly, irrespective of runtime noise (e.g., cache state, thread scheduling). Fixed PRNG seeds and canonical reduction orders ensure that honest executions always converge to the same result, preventing false slashing.

## 9.3 Privacy and Confidentiality

Confidentiality is preserved through threshold key management and TEE-based attestation (Section 5). Only attested enclaves executing approved containers ever reconstruct $\mathsf{sk}_{\mathsf{app}}$. All other components—including operators, DA nodes, auditors, and even KMS shards—observe only cryptographic commitments or encrypted payloads. Thus, verifiability and confidentiality reinforce one another: verification speaks to correctness, while TEEs guarantee that verification does not leak sensitive user data.

## 9.4 Liveness and Fault Tolerance

EigenDA ensures that ciphertexts and receipts remain retrievable for the duration of the challenge window. EigenVerify's committee sampling tolerates partial failures: if some verifiers are offline or unresponsive, the remaining honest majority can still reach a verdict. Timeouts ensure that the system progresses even if no challenge is raised, providing liveness equivalent to other optimistic systems.

Table 7: Residual risks and planned mitigations.

| Residual Risk | Mitigation / Roadmap |
|---|---|
| Cross-architecture portability | Maintain per-architecture verifier pools; explore numeric normalization for cross-device verification. |
| Closed-source library paths | Replace remaining cuBLAS/cuDNN calls with open deterministic kernels. |
| Economic parameter drift | Regular governance calibration and dynamic fee markets. |
| Operator cartelization | Stake decentralization; randomized committee sampling. |

## 9.5 Residual Risks and Mitigations

Table 7 summarizes remaining risks. Some stem from hardware trust assumptions (TEEs), others from portability constraints (GPU architecture differences). In each case, we outline roadmap items to further reduce exposure.

Overall, EigenAI achieves layered, composable security: determinism provides technical reproducibility, TEEs enforce confidentiality, EigenDA guarantees availability, and EigenLayer adds cryptoeconomic correctness. These layers interlock to produce a verifiable inference system resilient to both adversarial behavior and accidental faults.

# 10 Evaluation and Experiments

We evaluate EigenAI along three axes: (i) the robustness of bit-exact determinism under realistic deployment conditions, (ii) the performance overhead of deterministic kernels relative to vendor-optimized baselines, and (iii) the end-to-end cost of verification in light and full challenge scenarios. All experiments were conducted on NVIDIA H100 GPUs using pinned container digests and identical software environments.

## 10.1 Determinism Verification

We first assess whether EigenAI's execution stack produces bit-identical outputs across repeated runs and heterogeneous deployment settings. Repeated inference on the same host yielded perfect equality across all logits and generated tokens. Cross-host experiments—running identical containers on two independent H100 nodes—also produced exact matches. To probe sensitivity to batching and runtime variability, we perturb the batch size by ±20%, observing no divergence. As expected, cross-architecture tests (A100 versus H100) do not match bitwise due to differences in floating-point rounding behavior, underscoring the need for per-architecture verifier sets.

Table 8: Determinism evaluation across hosts and configurations.

| Configuration | Hosts | Batches | Match |
|---|---|---|---|
| Same host | 1 | 10 | 100.0% |
| Different hosts / same SKU | 2 | 10 | 100.0% |
| Batch-size variance ±20% | 2 | 20 | 100.0% |
| Diff arch (A100 vs H100) | 2 | 10 | 0.0% |

Table 9: Throughput and latency comparison (batch = 8, seq = 1024).

| Kernel Type | Rel. Thrpt | Overhead |
|---|---|---|
| cuBLAS (baseline) | 1.00× | – |
| Det. GEMM (EigenAI) | 0.97× | +2.4% |
| Det. mixed-precision | 0.95× | +4.1% |
| End-to-end LLM inference | 0.98× | +1.8% |

## 10.2 Stress and Batch-Invariance Tests

To evaluate robustness under operational noise, we co-schedule background GPU workloads that induce synthetic jitter and scheduling variability. Despite this perturbation, all runs produced identical outputs, confirming that deterministic kernel design—warp-synchronous reductions, fixed decoding order, and pinned software stack—effectively isolates inference from transient runtime effects. These results indicate that EigenAI's determinism holds not only under idealized conditions but also in realistic multi-tenant and performance-variable environments.

## 10.3 Performance Overhead

Next, we quantify the throughput and latency cost of deterministic kernels relative to vendor-optimized baselines. On Hopper GPUs, our deterministic GEMM kernels achieve 97–99% of cuBLAS throughput for quantized matrix multiplications, and approximately 95% for mixed-precision projection layers. End-to-end LLM inference shows only a modest latency increase ($\approx 1.8\%$), demonstrating that determinism can be achieved without compromising state-of-the-art performance.

# 11 Limitations and Future Work

Although EigenAI achieves deterministic inference and robust verification, several open challenges remain:

- **Cross-Architecture Reproducibility.** Determinism currently holds only within fixed GPU families. Future work includes portable numeric normalization to enable heterogeneous verifier sets.
- **Residual Library Paths.** Certain cuBLAS and cuDNN operations remain closed-source; we plan to replace them with open deterministic equivalents to achieve complete auditability.
- **Tool and API Determinism.** Agents that call external APIs or tools require deterministic transcript recording; EigenAI will extend receipts to include signed external call logs.

## 12 Conclusion

EigenAI unites deterministic GPU inference, privacy-preserving verification, and EigenLayer's cryptoeconomic guarantees into a single coherent platform. By making AI results reproducible, auditable, and slashable under fraud, it delivers a practical route to **verifiable AI at state-of-the-art performance**. These properties enable trustworthy *sovereign agents*—AI systems that can autonomously act, reason, and transact across high-stakes domains both on- and off-chain. As deterministic computation and cryptoeconomic security converge, verifiable intelligence becomes a first-class primitive for decentralized and enterprise ecosystems alike.

## References

[1] Ghodsi, Z., Gu, T., and Garg, S. "SafetyNets: Verifiable Execution of Deep Neural Networks on an Untrusted Cloud." *Advances in Neural Information Processing Systems 30*, 2017.

[2] Kang, D., Hashimoto, T., Stoica, I., and Sun, Y. "Scaling Up Trustless DNN Inference with Zero-Knowledge Proofs." arXiv:2210.08674, 2022.

[3] Gal, Y., and Ghahramani, Z. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." *Proc. 33rd ICML*, 2016.

[4] Lakshminarayanan, B., Pritzel, A., and Blundell, C. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles." *NeurIPS 30*, 2017.

[5] Wang, X., Wei, J., Schuurmans, D., *et al.* "Self-Consistency Improves Chain-of-Thought Reasoning in Language Models." *Proc. ICLR 2023.*

[6] Atıl, B., Aykent, S., Chittams, A., *et al.* "Non-Determinism of 'Deterministic' LLM Settings." arXiv:2408.04667 (v2 Apr 2025).

[7] PyTorch Core Team. "Reproducibility—Notes on Randomness and Determinism." *PyTorch Documentation v2.7*, 2023.

[8] ONNX Runtime Team. "Performance Tips for ONNX Runtime Web (WASM Backend)." Technical note, 2023.

[9] NVIDIA Corporation. *cuBLAS Library User Guide, Release 12.3*, §2.1 "Results Reproducibility." Santa Clara, CA, Aug 2023.

[10] NVIDIA Corporation. *cuBLAS Library User Guide*, CUDA Toolkit v12.9, §2.4.20, 2023.

[11] Shanmugavelu, A., *et al.* "Impacts of Floating-Point Non-Associativity on Reproducibility for HPC and Deep Learning Applications." arXiv:2403.11545, 2024.

[12] Coleman, C., and Siemons, J. "Non-Determinism in GPU-Accelerated Deep Learning Frameworks." *arXiv:2208.13040*, 2022.

[13] EigenLabs. *EigenVerify Overview (Objective Dispute Resolution Preview).* EigenCloud Documentation, 2025.

[14] EigenLabs. *EigenCloud Brings Verifiable AI to Mass Market with EigenAI and EigenCompute Launches.* EigenCloud Blog, Sept 2025.

[15] NVIDIA Corporation. *CUDA Compatibility Guide for Developers*, 2023.

[16] NVIDIA Corporation. *NVIDIA System Management Interface (nvidia-smi) User Guide*, 2023.

[17] NVIDIA Corporation. "Best Practices for NVIDIA Container Images." Technical documentation, rev. 2024.

[18] EigenLabs. *EigenLayer Core Protocol and Restaking Architecture.* Technical white paper, 2025.

[19] Shamir, A. "How to Share a Secret." *Communications of the ACM*, 22(11):612–613, 1979.

[20] Intel Corporation. *Intel Software Guard Extensions (SGX) Developer Guide*, 2016.

[21] AMD. *Secure Encrypted Virtualization (SEV) Architecture Reference Manual*, rev. 1.55, 2020.

[22] Murphy, S. "Building CUDA Images on GitHub Runners with Nix." Technical note, 2024.

[23] NVIDIA. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.* GitHub repository, accessed May 2025.

[24] Stoyanov, R., *et al.* "CRIUgpu: Transparent Checkpointing of GPU-Accelerated Workloads." arXiv:2502.16631, 2025.

[25] NVIDIA Developer Blog. "Reproducible Results with cuBLASLt." June 2023.

[26] NVIDIA Developer Blog. "Using CUDA Warp-Level Primitives." July 2018.

[27] Riach, D. "Determinism in Deep Learning." NVIDIA GTC Presentation S9911, 2019.

[28] EigenLabs. *EigenLayer Economic Security and Slashing Parameters.* Technical whitepaper, 2025.

[29] EigenLabs. *EigenLayer Governance Framework.* Documentation, 2025.

[30] EigenLabs. *EigenDA: Data Availability for Verifiable Compute.* Whitepaper, 2025.

[31] EigenLabs. *EigenCompute: Verifiable Container Runtime for Deterministic Agents.* Technical documentation, 2025.

[32] EigenLabs Research. *Empirical Evaluation of Deterministic Inference Kernels on Hopper GPUs.* Internal report, 2025.

[33] Zhang, Y., and Coleman, C. "DetBench: Benchmarking Deterministic Deep Learning Kernels." *arXiv:2411.01854*, 2024.

[34] Blackman, D., and Vigna, S. "Scrambled Linear Pseudorandom Number Generators." *ACM Transactions on Mathematical Software*, 45(2):28, 2018.

[35] Docker Inc. *Content Addressable Storage and Image Digests in Docker.* Technical documentation, 2023.

[36] EigenLabs. *EigenDA Proof Structures and Merkle Verification API.* Developer documentation, 2025.