PeerAI

A Cryptographic-Native, Settlement-Agnostic Protocol for **Decentralised AI Inference**

VERSION	DATE	STATUS
2.0	April 2026	Living Document

This document describes the protocol as implemented through Phase C (April 2026).

Abstract

We present PeerAI, a peer-to-peer protocol for AI inference that derives all system trust from cryptographic primitives rather than from any blockchain or trusted third party. The protocol introduces three novel constructions: (1) a *self-verifiable proof of inference* -- a signed execution receipt that can be verified offline with only the producing node's public key; (2) a *gossip-based Merkle reputation tree* -- a provable reputation history that requires no on-chain anchoring to function; and (3) a *settlement-agnostic escrow interface* -- a typed abstraction layer that allows any payment mechanism, from signed receipts to payment channels to smart contracts on any chain, to be selected at configuration time without code changes. Additionally, the protocol provides a *pluggable storage layer* over which encrypted session context can be persisted on local filesystems, IPFS, Walrus, or any content-addressed store. Together, these primitives produce a system that operates with full cryptographic integrity with zero blockchain dependencies, while making any chain an optional enhancement rather than a requirement.

Table of Contents

D. Appendix D: Reputation Score Parameterisation

1. Introduction

AI inference has consolidated into a small number of centralised API providers. This consolidation creates structural dependencies that are problematic at every layer of the stack: the infrastructure is controlled by a single party, the data is accessible to that party, pricing is determined unilaterally, and access can be revoked without notice or recourse.

The proposed remedy -- running models locally -- is infeasible for most users. The frontier models that produce state-of-the-art outputs require hundreds of gigabytes of GPU memory, which is economically and practically inaccessible outside institutional settings.

The logical alternative is a *marketplace*: users who need more compute than they own pay users who have spare GPU capacity. This idea is not new. What is new is how trust is established in such a marketplace. Every prior system grounds trust in one of two ways: either a trusted coordinator (which reintroduces centralisation) or a specific blockchain (which imposes a hard dependency on that blockchain's availability, token economy, and ecosystem).

PeerAI takes a third path: trust is grounded exclusively in cryptography. A node's identity is an Ed25519 keypair. Its track record is a Merkle tree of signed execution receipts. Any client can verify any receipt with only the node's public key -- no network call, no blockchain, no trusted party. Payment can then be settled on any chain, through off-chain channels, or not at all -- the cryptographic layer functions identically regardless of which settlement mechanism is chosen.

This paper describes the protocol in detail. Section 5 defines the cryptographic primitives. Sections 6-9 describe the four novel layers (proof, reputation, storage, settlement). Sections 10-12 describe the marketplace and session privacy. Sections 13-14 analyse the economics and security. Section 15 compares the design with prior work.

2. Problem Statement and Motivation

2.1 The Centralisation Failure Mode

Let P be a user prompt, R be the provider's response, and C be the conversation context accumulated over a session. In current cloud AI systems:

- The provider observes (P, C, R) for every turn.

- The provider controls whether the user may access the service.

- The provider sets the price unilaterally.

- If the provider's infrastructure fails, no inference occurs.

None of these are technical requirements. They are consequences of the architecture, not the problem.

2.2 What a Decentralised Inference Network Must Provide

A viable decentralised alternative must provide five guarantees that cloud AI does not:

G1 -- Session Privacy	The GPU node performing inference must not be able to learn the user's accumulated conversation context C beyond what is necessary for the immediate inference turn.
G2 -- Node Accountability	After a job completes, any third party must be able to verify that node N performed job J with the claimed parameters, without trusting node N or any coordinator.
G3 -- Settlement Neutrality	The protocol must not require a specific token, chain, or wallet infrastructure to function. Settlement is a mechanism for transferring value; it must be separable from the mechanisms for routing, inference, and trust.
G4 -- Storage Neutrality	Persistent session context must not be tied to any specific storage provider. The user must be able to retrieve their session from any supported backend.
G5 -- Permissionless Participation	Any node that satisfies the protocol requirements must be able to join and earn, without approval from a central authority.

2.3 Why Existing Systems Fail to Provide These Guarantees

A systematic review appears in Section 15. In summary: every prior system either lacks G2 (no verifiable accountability) or sacrifices G3/G4 (hard-coded to a specific chain and storage system). No existing system provides all five guarantees simultaneously.

3. Related Work

3.1 Bittensor

Bittensor [Rao and Tanner, 2021] is a decentralised ML network that uses its own Substrate-based blockchain and the TAO token as the settlement and reputation mechanism. Validators score miner responses and emissions are distributed proportionally. The system is entirely dependent on the TAO ecosystem: running a subnet without TAO infrastructure is not possible. Guarantees G3 and G4 are violated. Session privacy (G1) is not provided -- validators observe all queries.

3.2 io.net and Akash Network

io.net [2024] and Akash [Akash Network, 2020] both provide decentralised compute marketplaces. However, they are optimised for batch training workloads, not real-time conversational inference. Neither provides session continuity, end-to-end encryption of conversation context, or a per-request payment mechanism suitable for inference latencies of 100-500 ms. They are compute clouds, not inference protocols.

3.3 QVAC (Tether)

QVAC [Tether, 2024] is a true P2P inference protocol using the Holepunch/Hyperswarm networking stack. It provides permissionless participation but has no payment mechanism, no reputation system, and no session privacy. Without economic incentives, there is no rational reason for operators to provide GPU capacity to strangers. G2 and G5 are satisfied; G1, G3, and G4 are not addressed.

3.4 Fortytwo

Fortytwo is a quality-focused inference router that uses Bradley-Terry scoring to select the highest-quality response from multiple model backends. It operates as a centralised coordinator, violating G5. Its session model is single-shot question-and-answer, not persistent conversation. It has no P2P layer, no session encryption, and no on-chain settlement.

3.5 Summary

No prior system provides all five guarantees. PeerAI is the first protocol to do so.

4. System Overview

PeerAI is structured as a six-layer stack. Each layer is independently replaceable. The layers interact only through well-defined trait interfaces; no layer reaches through another layer.

Layer 6	**Application**	TypeScript SDK, OpenAI-compatible HTTP API, Web UI
Layer 5	**Session**	End-to-end encrypted portable conversation memory
Layer 4	**Reputation**	Merkle tree of signed proofs, gossip-based
Layer 3	**Marketplace**	P2P request broadcast, bid collection, job dispatch
Layer 2	**Settlement**	Pluggable: free / signed receipt / channel / chain
Layer 1	**Storage**	Pluggable: local / IPFS / Walrus / Arweave
Layer 0	**Cryptography**	Ed25519 identity, SHA-256 content addressing, AES-256-GCM

Layer 0 has no external dependencies. Every layer above it may optionally use external infrastructure, but none is required for the protocol to function. The key architectural decision is that Layer 2 (Settlement) and Layer 1 (Storage) are configuration choices, not protocol requirements. A node operator selects which backends to activate by editing a TOML configuration file. The node binary is identical in all configurations.

5. Cryptographic Foundation

5.1 Node Identity

Every node in the PeerAI network is identified by an Ed25519 keypair (sk_N, pk_N) where sk_N is the secret signing key (256 bits), generated locally and never transmitted, and pk_N is the corresponding public key on the Edwards25519 curve.

The node's network identity (its libp2p PeerId) is derived from pk_N by hashing it through the identity multicodec. This means a node's network address and its cryptographic identity are the same object -- there is no separate account or wallet address required to participate.

5.2 Content Addressing

All stored blobs are identified by their SHA-256 digest: id(b) := SHA-256(b). This property means:

- The same content always has the same ID -- deduplication is free.

- The ID commits to the content -- any modification changes the ID.

- IDs are backend-agnostic -- the same blob on IPFS and on the local filesystem share the same identifier.

For IPFS, the CID (Content Identifier) is computed differently (using IPFS's multihash standard), but the semantic property is identical: the CID is a cryptographic commitment to the content.

5.3 Session Encryption

Session context is encrypted using authenticated symmetric encryption. The scheme operates as follows:

Key Generation: A SessionKey is derived from a fresh X25519 Diffie-Hellman exchange between the client's ephemeral keypair and a static identity. In practice, the key is generated locally and never leaves the client device.

Encryption: Given plaintext m and key K:

```
nonce  := random(96 bits)
(c, t) := AES-256-GCM(K, nonce, m)
blob   := nonce || t || c

where t is the 128-bit authentication tag and c is the ciphertext.
```

Decryption: Given blob and K, the nonce, tag, and ciphertext are parsed and decryption proceeds via AES-256-GCM.Decrypt. If the authentication tag does not verify, decryption returns an error. The GPU node never observes K. It receives only the encrypted blob and the nonce, computes inference on the decrypted context in memory briefly, and the plaintext is zeroed from RAM after use.

6. The ProofOfInference Primitive

6.1 Definition

A ProofOfInference is a signed execution receipt. Formally, it is a tuple:

```
pi := (req_id, sess_id, peer_id, client, model,
       t_in, t_out, delta, price, ts,
       H_in, H_out, settle_id, escrow_tx,
       pk_N, sigma)

where:
  req_id, sess_id  -- unique identifiers for this request and session (UUIDs)
  peer_id          -- the node's libp2p PeerId string
  client           -- the client's address or public key identifier
  model            -- the model identifier string
  t_in, t_out      -- input and output token counts
  delta            -- wall-clock latency in milliseconds
  price            -- amount paid in NanoX (1 X = 10^9 NanoX)
  ts               -- Unix timestamp
  H_in             -- SHA-256(encrypted_prompt || context_blob_id)
  H_out            -- SHA-256(response_tokens)
  settle_id        -- settlement adapter used
  escrow_tx        -- optional on-chain transaction ID
  pk_N             -- the node's Ed25519 public key (32 bytes)
  sigma            -- the node's Ed25519 signature (64 bytes)
```

6.2 Canonical Serialisation

To sign the proof, the node produces a canonical byte representation of all fields except the signature: canonical(pi) := JSON-encode(pi \ {sigma}). JSON is used for canonical serialisation because field ordering is determined by declaration order in the struct, and all field types have unambiguous JSON representations. This makes the format cross-language verifiable.

6.3 Signing

At job completion, the node computes: sigma := EdDSA.Sign(sk_N, canonical(pi)), where EdDSA.Sign is Ed25519 signing as defined in RFC 8032. The signature is 64 bytes.

6.4 Verification

Anyone holding the proof can verify it by recovering the verifying key from pk_N, then calling EdDSA.Verify with the canonical message and signature. This verification requires only the proof bytes and knowledge of the Ed25519 scheme -- no network call, no blockchain query, no trusted third party. The proof is self-contained.

6.5 Security Properties

Binding: The signature commits to all fields. Changing any field (inflating t_out, reducing delta, changing model) invalidates the signature.

Non-repudiation: A node cannot later deny having produced a proof because the signature can only be produced by the holder of sk_N.

Tamper Detection: Because H_in and H_out are included in the signed fields, any discrepancy between the claimed output hash and the actual response bytes can be detected by any third party.

7. The Reputation System

7.1 Motivation

The naive reputation model -- a floating-point score stored in a database -- has two fatal flaws: the database operator can manipulate it, and there is no way to audit the basis for any particular score. PeerAI replaces this with a data structure whose root is a single 32-byte hash and whose membership proofs can be verified cryptographically.

7.2 The Merkle Reputation Tree

Each GPU node maintains a Merkle tree of its completed job proofs. Let H := SHA-256 and let {pi_1, pi_2, ..., pi_n} be the ordered sequence of proofs produced by node N. The leaf hashes are computed as: l_i := id(pi_i) = H(canonical(pi_i)) for i = 1..n.

The tree is constructed bottom-up. If the number of leaves is odd, the last leaf is duplicated. Internal nodes are computed as: node(L, R) := H(L || R). The root is: root(T_N) := build_up({l_1, ..., l_n}).

7.3 Gossip Protocol

Every ten minutes, node N broadcasts a NodeAnnouncement over the libp2p gossipsub reputation topic. The announcement contains (peer_id, root(T_N), ts) where ts is a timestamp to prevent replay. Any network participant who receives this announcement knows that node N claims its proof history hashes to root(T_N), and may request individual proofs and verify their membership with a Merkle path.

7.4 Reputation Scoring Function

Given a verified set of proofs V(N), the reputation score of node N is:

```
score(N) := alpha * success_rate(N) + beta * latency_score(N)

success_rate(N)  := |V(N)| / |T_N|
avg_latency(N)   := (1/|V(N)|) * sum(pi.delta for pi in V(N))
latency_score(N) := max(0, 1 - avg_latency(N) / L_max)

with alpha = 0.6, beta = 0.4, and L_max = 5000 ms
```

7.5 Optional Chain Anchoring

Periodically (default: once per hour), a node may anchor root(T_N) on-chain. This produces an immutable on-chain timestamp: anyone who holds root(T_N) can verify it was published no later than the timestamp of the anchor transaction. The cost is one transaction per hour regardless of how many proofs have been added since the last anchor. This is an optional enhancement -- the reputation system functions identically without chain anchoring.

8. The Storage Layer

8.1 Design Philosophy

Session context C is a structured blob of conversation history. It must be persisted across turns, across devices, and optionally across time. The key constraint is that the blob must be: (1) content-addressed -- the identifier cryptographically commits to the content; (2) backend-agnostic -- the same protocol must work whether the blob is stored locally, on IPFS, on Walrus, or any other backend; and (3) client-keyed -- only the client who holds the session key K can read the blob.

8.2 The StorageClient Interface

```
StorageClient :=
  put(data: bytes, ttl_epochs: u64) -> BlobId
  get(blob_id: BlobId)              -> bytes
  delete(blob_id: BlobId)           -> ()
```

8.3 Backend Comparison

Backend	ID Scheme	Dedup?	Permanent?	Dependencies
LocalFile	SHA-256 hex	Yes	Until deleted	None
IPFS	CIDv0/v1	Yes (network-wide)	Until unpinned	IPFS node / pinning service
Walrus	Walrus blob ID	Partial	TTL in epochs	WAL token, Walrus network
Memory	Counter	No	Process lifetime	None (test-only)

8.4 Session Storage Flow

```
# Per turn:
C_k    := append(C_{k-1}, (user_prompt, assistant_response))
blob_k := AES-256-GCM.Encrypt(K, C_k)
id_k   := storage.put(blob_k)

# Session recovery on a new device:
blob_k := storage.get(id_k)
C_k    := AES-256-GCM.Decrypt(K, blob_k)
# No account, no server, no network query -- just K and id_k.
```

9. The Settlement Layer

9.1 Design Philosophy

Payment is a mechanism for transferring value in exchange for a completed service. It does not need to be a blockchain operation. The choice of payment mechanism should be orthogonal to the choice of inference routing, session privacy, and reputation. PeerAI defines a SettlementAdapter interface that abstracts over the full space of payment mechanisms, from no payment at all to full on-chain escrow.

9.2 The SettlementAdapter Interface

```
SettlementAdapter :=
  id()                                    -> str
  capabilities()                          -> SettlementCapabilities
  lock_funds(p: EscrowParams)             -> EscrowHandle
  release_funds(h: EscrowHandle, pi: Proof) -> ()
  refund_funds(h: EscrowHandle)           -> ()
  get_balance(addr: str)                  -> NanoX
  anchor_hash(hash: [u8;32], label: str)  -> Option<str>
```

9.3 Settlement Adapters

Free Settlement: FreeSettlement is a complete no-op. All methods succeed without performing any action. Despite being free, jobs still produce signed ProofOfInference receipts and reputation tracking works identically.

Signed Receipt: Provides lightweight accountability without on-chain infrastructure. The node performs work, signs the proof, and sends it to the client. Appropriate for micro-payments and networks with mutual reputation.

Payment Channels: Allow a client and node to transact many times off-chain. The economic effect is full escrow but with gas costs amortised over n requests (2g/n per request vs g per request for on-chain escrow).

On-Chain Escrow (Sui, EVM): For maximum trustlessness, the system supports smart contract escrow on any chain. The escrow contract verifies the ProofOfInference on-chain before releasing funds.

9.4 Settlement Compatibility Matching

A node N has an ordered list of settlement adapters. A client C has an ordered list of accepted settlement IDs. When the client receives a bid from node N, the bid is compatible if there exists an offer in the bid whose settlement_id appears in the client's accepted list. The selected settlement adapter is the first match in the node's preference order. A node that only accepts Sui escrow and a client that only has an Ethereum wallet simply do not match -- but they coexist on the same P2P network without conflict.

10. The P2P Marketplace

10.1 Network Layer

The PeerAI P2P network is built on libp2p with the following protocol stack:

Protocol	Purpose
TCP + QUIC	Transport (dual-stack)
Noise	Authenticated encryption at the transport layer
Yamux	Stream multiplexing
Kademlia DHT	Peer routing and discovery
mDNS	Local network discovery
Gossipsub	Publish-subscribe for marketplace and reputation messages
Identify	Protocol negotiation and address exchange
AutoNAT	NAT traversal classification
Ping	Liveness

10.2 Gossipsub Topics

Topic	Publisher	Subscriber	Content
/peerai/inference/any/1.0.0	Clients	All GPU nodes	InferenceRequest
/peerai/inference/{model}/1.0.0	Clients	Model-specific GPU nodes	InferenceRequest
/peerai/bids/1.0.0	GPU nodes	Clients	InferenceBid
/peerai/announce/1.0.0	All nodes	All nodes	NodeCapabilities
/peerai/reputation/1.0.0	All nodes	All nodes	Merkle root announcement

10.3 The Auction Protocol

The inference marketplace operates as a sealed-bid first-price auction with a 200 ms collection window:

1. Client broadcasts InferenceRequest on the inference topic for the requested model.

2. GPU nodes that satisfy all six bid criteria broadcast InferenceBid on the bids topic.

3. Client collects bids for 200 ms, filters to compatible bids, and selects the winner by composite score: score(bid) := w1*(1/bid.price) + w2*(1/bid.latency) + w3*bid.reputation, with w1=0.4, w2=0.3, w3=0.3.

4. Client sends the job directly to the winning node's API endpoint.

5. Node executes inference and streams the result back.

6. On completion, node signs ProofOfInference and sends it to the client.
7. Node records the proof in its local Merkle tree.
8. Settlement proceeds according to the selected adapter.

11. Session Privacy

11.1 The Privacy Model

A GPU node learns only the content of the current inference turn's decrypted prompt and context window. It does not

This guarantee is achieved by the session architecture. The user's full conversation history C is stored as an encrypted blob. Before each turn, the client: (1) fetches the encrypted blob; (2) decrypts it with K (client-side only -- K never leaves the device); (3) constructs the context window W containing recent messages that fit in the model's context; (4) encrypts W for the specific GPU node using the node's public key via X25519 DH exchange; and (5) sends the encrypted context window with the inference request.

11.2 Privacy Levels

Level	Description	Additional Cost
Standard	Context window encrypted and transmitted to one node	None
Private	Context window encrypted; node cannot cache between turns	Slight latency
Fragmented	Prompt fragmented across multiple nodes via DIP	2-3x latency
Maximum	Fragmentation + TEE (Trusted Execution Environment) required	3-5x latency

12. The Bid Matching Protocol

12.1 Node-Side Bid Decision

A GPU node receives an InferenceRequest and evaluates six criteria in sequence (cheapest to most expensive):

1. Model Check	req.model is in node.available_models
2. Capacity Check	node.active_jobs < node.max_concurrent_jobs
3. Queue Depth	node.queue_depth <= 2 x node.max_concurrent_jobs
4. Economic Check	req.budget >= node.min_price x 100 / 1000
5. Privacy Check	req.privacy != Maximum OR node.tee_enabled
6. Throttle Check	node.wins_last_60s < 4 x node.concurrent_jobs

12.2 Client-Side Bid Selection

The client collects all compatible bids and selects the winner by composite scoring with weights w1=0.4 (price), w2=0.3 (latency), and w3=0.3 (reputation). Ties are broken by lowest price. The bid includes the node's accepted_settlements list, making the bid self-describing: any client can determine from the bid alone whether a match is possible, without querying any external registry.

13. Economic Analysis

13.1 Node Incentives

A GPU node's expected revenue is a product of the expected number of jobs won and the expected price per job. The bid win probability is a function of price, latency, and reputation. A node that serves higher quality inference at lower latency with a better reputation has a higher win probability, creating a competitive market for quality.

13.2 Client Incentives

Clients benefit from price discovery via open auction, model diversity across different nodes, censorship resistance (no single node can block a client), and privacy (session key never leaves the client device).

13.3 Payment Channel Economics

Payment channels dramatically reduce the cost of high-frequency inference. If g is the gas cost of an on-chain transaction and a user performs n inference requests per session:

```
cost_per_request(on-chain escrow) = g          (per request)
cost_per_request(payment channel) = 2g / n     (amortised over n requests)

For n = 100 requests: amortised cost = 0.02g per request -- a 50x reduction.
```

For micro-transactions (below $0.01 per request), payment channels or signed receipts are the economically rational choice. For high-value requests (above $0.10), on-chain escrow provides stronger guarantees that justify the gas cost.

13.4 Free Mode and the Social Contract

FreeSettlement supports shared GPU resources within a company or research group, personal nodes serving only the owner's queries, and bootstrapping a new network before token infrastructure exists. In free mode, the reputation system still functions -- nodes build verifiable track records even without collecting payment. When the operator later activates a paid settlement adapter, their established reputation provides a competitive advantage in the marketplace.

14. Security Analysis

14.1 Threat Model

The analysis considers adversaries who may control a fraction f < 1/2 of GPU nodes, observe all messages on the P2P network, and attempt to forge proofs, inflate reputation, or steal client funds.

14.2 Proof Forgery

An adversary cannot forge a ProofOfInference for a node N without access to sk_N. The proof contains a signature produced by EdDSA.Sign(sk_N, canonical(pi)). Forging a valid proof requires breaking the Ed25519 signature scheme, which reduces to the hardness of the discrete logarithm problem on the Edwards25519 curve, believed to provide 128 bits of security.

14.3 Reputation Inflation

A node cannot inflate its reputation score without a colluding client's signature. Every proof in the Merkle tree is independently verified. A tampered proof (with inflated token counts or reduced latency) fails signature verification and reduces rather than inflates the reputation score. Collusion is mitigated by the real GPU cost of running fake jobs, rate-limiting via the throttle check, and anomaly detection of implausible proof patterns.

14.4 Escrow Safety

For on-chain adapters, escrow safety reduces to the security of the smart contract implementation. The release_funds method calls the contract only if pi.verify() passes. The contract independently re-verifies the proof before releasing funds, ensuring that even a malicious node binary cannot trigger a release without a valid proof.

14.5 Session Privacy Under Semi-Honest Nodes

GPU nodes are modelled as semi-honest: they follow the protocol but may attempt to extract information from what they observe. Under this model, the node observes only the decrypted context window W for the duration of the inference turn, and does not observe the pruned history or the session key K. After the turn, if memory wiping is enabled, W is zeroed from RAM.

14.6 Network-Level Attacks

Because the P2P network uses Noise-encrypted connections, network observers cannot read message content. They can observe the graph structure but not the content of any message. For higher anonymity, the Fragmented and Maximum privacy levels distribute the query across multiple nodes.

15. Comparison with Prior Systems

15.1 Feature Matrix

Property	PeerAI	Bittensor	QVAC	io.net	Fortytwo
G1 -- Session Privacy	Yes	No	No	N/A	No
G2 -- Node Accountability	Yes	Partial	No	No	No
G3 -- Settlement Neutrality	Yes	No	Yes (no payment)	No	N/A
G4 -- Storage Neutrality	Yes	No	No	No	N/A
G5 -- Permissionless	Yes	Yes	Yes	No	No
Real-time Inference	Yes	Yes	Yes	No	Yes
Persistent Sessions	Yes	No	No	No	No
Streaming Responses	Yes	No	No	No	No
Multi-chain Settlement	Yes	No	N/A	No	N/A

15.2 The Key Differentiator

The single most important differentiator is that PeerAI is the only system in which the source of trust is the cryptographic primitive -- not the blockchain, not the network, not a trusted coordinator. Everything else follows from this choice.

Bittensor grounds trust in the TAO token and its validators. Remove the TAO infrastructure and the trust model collapses. QVAC grounds trust in nothing -- it has no economic model. Fortytwo grounds trust in its coordinator backend. PeerAI grounds trust in Ed25519 signatures and SHA-256 Merkle proofs. Remove any specific blockchain and the system continues to work with identical cryptographic guarantees. Add a blockchain and it becomes an optional anchor for an already-sound system.

16. Conclusion

We have presented PeerAI, a protocol that achieves all five of the guarantees identified as necessary for a viable decentralised AI inference network. The key insight is the separation between the cryptographic layer (where trust is established) and the settlement layer (where value is transferred). By making the cryptographic layer self-sufficient and treating the settlement layer as pluggable infrastructure, the protocol:

- Works with zero blockchain dependencies in its simplest configuration
- Can adopt any blockchain as an optional settlement and anchoring layer
- Provides verifiable node accountability through signed execution receipts
- Provides verifiable reputation through gossip-based Merkle proofs
- Provides session privacy through client-held encryption keys and pluggable storage

The storage layer and settlement layer are independently configurable. A user on an air-gapped network can use local storage and free settlement; a user on a public network can use IPFS and payment channels; an enterprise deployment can use Walrus and Sui escrow. The protocol's behaviour is identical in all configurations because the trust model is identical: Ed25519 signatures and SHA-256 Merkle proofs.

The three constructions introduced here -- the self-verifiable proof of inference, the gossip-based Merkle reputation tree, and the settlement-agnostic escrow interface -- are general enough to be applicable beyond AI inference. Any protocol that requires verifiable accountability for off-chain computation and pluggable settlement could adopt these constructions directly.

A. Appendix A: Proof of Inference Verification Algorithm

```
Input:  pi = (req_id, sess_id, peer_id, client, model,
              t_in, t_out, delta, price, ts,
              H_in, H_out, settle_id, escrow_tx,
              pk_N, sigma)

Steps:
  1. msg  := JSON-encode(pi \ {sigma})      -- canonical bytes
  2. vk   := EdDSA.VerifyingKey(pk_N)        -- recover public key from bytes
  3. valid := EdDSA.Verify(vk, msg, sigma)   -- Ed25519 verification
  4. return valid

Complexity: O(1) -- constant time, no network calls
```

B. Appendix B: Merkle Tree Construction

```
Input:  proofs = [pi_1, pi_2, ..., pi_n]

Steps:
  1. leaves := [SHA-256(canonical(pi_i)) for i in 1..n]
  2. if n is odd: leaves.append(leaves[n-1])  -- duplicate last
  3. level := leaves
  4. while len(level) > 1:
       next := []
       for i in 0..len(level)-1 step 2:
         next.append(SHA-256(level[i] || level[i+1]))
       level := next
  5. return level[0]  -- root hash
```

C. Appendix C: Settlement Adapter Configuration Reference

```
# Free mode -- no payment required
[[settlement.adapters]]
id       = "free"

# Signed receipt -- off-chain, node signs proof
[[settlement.adapters]]
id           = "receipt"
price_per_1k = 5        # NanoX per 1000 output tokens

# Off-chain payment channel
[[settlement.adapters]]
id           = "channel"
price_per_1k = 8
token_id     = "native"

# Sui on-chain escrow (Phase D)
[[settlement.adapters]]
id               = "sui"
price_per_1k     = 10
rpc_url          = "https://fullnode.mainnet.sui.io"
package_id       = "0xABC..."
treasury_address = "0xDEF..."

# EVM on-chain escrow -- Base L2 (Phase E)
[[settlement.adapters]]
id               = "evm-8453"
price_per_1k     = 12
rpc_url          = "https://mainnet.base.org"
contract_address = "0x..."
token_address    = "0x..."
chain_id         = 8453
```

The adapters are evaluated in the order they appear in the configuration file. The first adapter whose id appears in the client's accepted_settlements list is selected for that job.

D. Appendix D: Reputation Score Parameterisation

Parameter	Default	Description
alpha	0.6	Weight of success rate in composite score
beta	0.4	Weight of latency score in composite score
L_max	5000 ms	Latency at which latency score contribution = 0
Gossip interval	600 s	How often Merkle root is broadcast
Anchor interval	3600 s	How often Merkle root is anchored on-chain

The composite score satisfies score(N) in [0, 1] for all valid inputs. A node with 100% success rate and average latency approaching 0 ms achieves score = 1.0. A node with 0% verified proofs achieves score = 0.0 regardless of

latency.

This document describes the protocol as implemented through Phase C (April 2026). Sections on Sui Settlement (Phase D), EVM Settlement (Phase E), on-chain payment channels (Phase F), and full gossip protocol (Phase G) describe planned functionality.